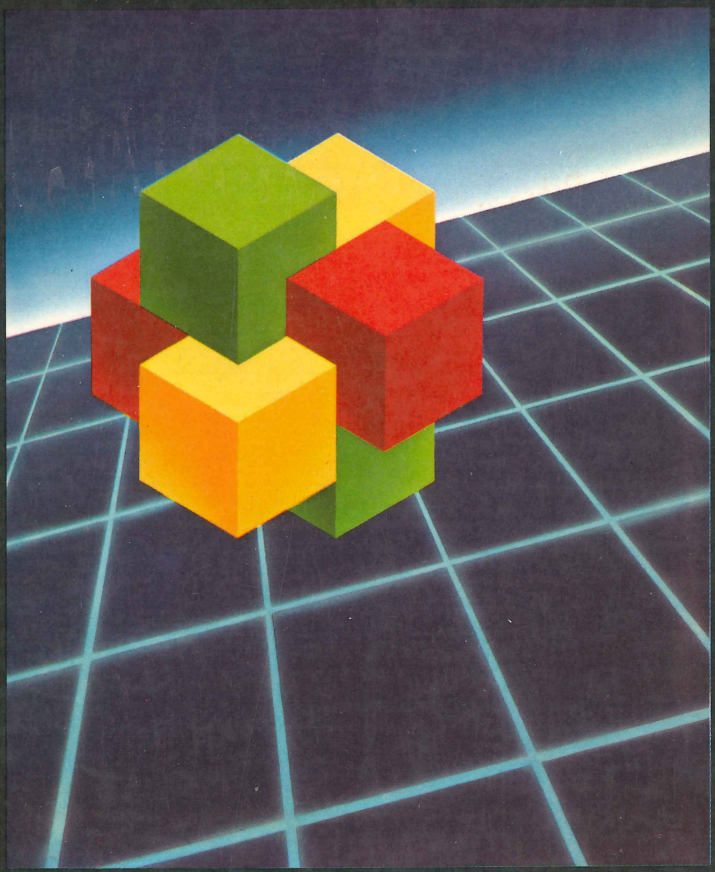
PANEL '85
EXPODATA

20 a 27 de Julho de 1985
UFRGS - Porto Alegre - Brasil

V Congresso da Sociedade Brasileira de Computação
XI Conferência Latino-Americana de Informática



ANAIS vol. I

XII SEMISH

MFN 814

005.74
CONgr
v.1.

PEDECIBA
INFORMATICA

V CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
XI CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA

Porto Alegre,Brasil, 20 a 27 de julho de 1985

A N A I S

Volume I

Editores: Ricardo A.L.Reis, Carlos E.Rech,Roberto M.J.Macedo

Promoção: SBC-Sociedade Brasileira de Computação
CLEI- Centro Latino-Americano de Estudos em Informática
UFRGS- Universidade Federal do Rio Grande do Sul

Patrocínio: Finep,Capes,CNPq,SEI,SERPRO,IFIP,UNESCO,IBI,CREI,FIPEC

Apoio: ITAUTEC,SCOPUS

24/10/88.

v. 1, 1985. 282

APRESENTAÇÃO

O XII Seminário Integrado de Software e Hardware, evento integrante do V Congresso da Sociedade Brasileira de Computação e XI Conferência Latino-Americana de Informática, apresenta uma síntese dos trabalhos científicos em Informática efetuados atualmente na América Latina.

Observamos uma evolução importante, na quantidade e na qualidade dos trabalhos submetidos, apesar dos graves problemas existentes quanto aos recursos destinados à pesquisa nos países latino-americanos. Entretanto a evolução é pequena se comparada com a evolução da Informática a nível mundial, no mesmo período.

A Sociedade Latino-Americana deve desenvolver urgentemente uma tecnologia própria em informática, como única maneira de resguardar a soberania de nossos países.

Este volume contém um tutorial, 61 artigos técnicos (incluindo os três trabalhos premiados no IV Concurso de Trabalhos de Iniciação Científica) e 10 comunicações técnicas. Estes trabalhos foram escolhidos, dentre 133 que foram submetidos ao XII SEMISH, sendo 115 artigos técnicos e 18 comunicações. A distribuição dos trabalhos submetidos, quanto ao país de origem, foi a seguinte:

PAÍSES	ARTIGOS TÉCNICOS	COMUNICAÇÕES
Argentina	11	3
Brasil	75	13
Chile	9	2
Equador	3	-
França	2	-
México	1	-
Uruguai	4	-
Venezuela	10	-

Ricardo Augusto da Luz Reis
Presidente do Comitê de Programa

P E D E C I B A
INFORMATICA

INDICE DOS TRABALHOS

SESSÃO 1 - REDES DE COMPUTADORES I

Aspectos da Implementação de um Protocolo de Transporte num Ambiente Seqüencial
P.R.F. Cunha, E.B. Lucena, J.A.S. Monteiro 1

SNPUC - Um Servidor de Nomes para a REDPUC
J.L. Medeiros, D.A. Menascé 11

Especificacion Formal de Protocolos de Comunicación
P.O' Callaghan, C. Rodrigues 36

Uma Implementação de um Protocolo de Transferência de Arquivos para Redes de Computadores
O. L. Tavares, D. Schwabe 41

Utilização de PROLOG para a Especificação e Validação de Serviços e Protocolos de Comunicação
W. L. de Souza 51

Técnicas de Multiplexação por Divisão no Tempo para Interconexão Processador Memória em Sistemas de Multiprocessadores: Um Estudo Comparativo
M.A.G. Brasileiro, J. A. B. Moura 60

SESSÃO 2 - COMPUTAÇÃO GRÁFICA

Entrada de Dados Gráficos em Sistema PAC
B.B. Sotomayor, C.L. Tozzi 72

Tipos de Datos Abstractos para la Especificacion de la Animación: Scripts, Actors y Movies
J.A. Fernandez, Andreas Meier 81

O Mapeamento de Textura na Síntese de Imagens Realistas
C.C. Alvarenga, R. N. Melo 92

Um Instrumento de Anālise de Imagens por Computador: A Morfologia Matemātica
C. Guichou, J. Barrera 105

O Projeto de um Terminal Grāfico de Mēdia Resolução
G.P. Azevedo, J.A.S. Borges 114

SESSÃO 3- INFORMĀTICA NA EDUCAÇÃO

Enseñanza asistida por Computador; La Enseñanza de La Programaciōn
J. Zambrano 121

SESSÃO 4- REDES DE COMPUTADORES II

Estudo Comparativo das Disciplinas Exaustiva e Não Exaustiva na Interliga ção de Redes Locais
D.A. Menasce, L. Lellis 132

Uma Rede Local em Anel para o DCC
A.A.F. Loureiro, D.C.S. Junior, J.E.R. Dantas, N.A.C. Lages 145

Software de Comunicação para uma Rede Local de Micros
L.F. Friedrich, S.S. Toscani, M. Tornquist 156

Annete: Una Red Local Econōmica de Alto Rendimiento para Microcomputadores
P. O'Callaghan , G. Broner 167

Transceptores para Redes Locais em Barra Comum
M.F. Vieira, J.N. Chame, A.M. Moreira, L.F.G. Soares 177

SESSÃO 5 - ARQUITETURAS NÃO CONVENCIONAIS

Proposta de uma Arquitetura Controlada pelos Dados
P. R. Foina 188

Arquitectura de Bus de Memoria Controlada por Dependência de Tareas
M. Nussbaum, A. Cipriano 201

A Arquitetura dos Supercomputadores
C. L. Amorin 211

SESSÃO 6 - INTELIGÊNCIA ARTIFICIAL

Sistema Especializavel de Reconhecimento de Conceitos
L. Fernando, J. Maia, P. Bezerra 222

Un Formalismo para Interrogar Base de Datos en Castellano
J. Lobo 230

SESSÃO 7 - GRAFOS E ALGORITMOS

Aspectos de Complexidade de Algoritmos Funcionais
S. Lemos Meira 243

Estudo Empírico de uma Nova Implementação para o Algoritmo de Construção de Árvore Patrícia
L.C.A. Albuquerque, N. Ziviani 254

Les Graphes Adjoints des Graphes Simples avec $\Delta(G) > 4$ sont Arête - Reconstructibles
O. J. M. Houttman 268

Um Ejercicio de Analisis para un Algoritmo de Sort
A. Pereda B., R. Contreras A. 281

Uma Rotina Didática para Quadratura Adaptativa
J. M. Marins 289

SESSÃO 8 - ENGENHARIA DE SOFTWARE

Metodologia para o Desenvolvimento de um Sistema de Aquisição de Dados e de Supervisão
J.M. Farines, R.S. de Lemos, L.F. Borges de Andrade 301

Uma Ferramenta para Geração de Diálogos
M.V. Silva, L.P. Magalhães 312

Metodologia de Construccion de Programas a partir de Especificaciones Formales
M. Rukoz 322

SODA - An Interactive Software Design Tool
A. M. de A. Price 331

A Linguagem Constructor para Ambientes Computacionais
P.F.W.K. de Buzin 345

Um Critério de Seleção de Casos Teste por Cobertura
A. Von Staa 354

SESSÃO 9 - SISTEMAS TOLERANTES A FALHAS

Um Microcomputador Tolerante a Falhas para Controle em Tempo Real
J.C. Barreto Leite, H.S. Castro 367

Circuitos Autotestáveis: Opção de Projeto de Circuitos visando o Teste Implícito ou Concorrente
I. Jansch 379

SESSÃO 10 - SISTEMAS DIGITAIS

Um Equipamento Automático de Testes para Manutenção e Qualificação
M.D.P. da Silveira 388

Desenvolvimento de um Analisador Espectral Utilizando FFT
H.P. Neves 394

Um Servidor de Impressão
M.H. Szwarcman, A. Nestorov 406

Traductor de un Language Estructurado a Descripcion Elemental de Hardware
H. A. Espina, R. Vallejos C. 411

SESSÃO 11 - SISTEMAS DISTRIBUIDOS

Suporte para Desenvolvimento de Sistemas Distribuídos Baseado num Barramento Paralelo Centralizado
C. Kinrner, E. Marques 423

Uma Arquitetura Flexível para Sistemas Distribuídos Tolerantes a Falhas
O.G.L. Filho 436

Um Modelo para a Concepção de Sistemas Distribuídos
P.F.W.K. de Buzin 447

O Kernel Distribuido Constructor
J. A. Queiróz 455

Modelizacion del Canal de Communicaciones de um Sistema Distribuido, en GPSS
M. de Giusti 464

SESSÃO 12 - LÓGICA E TEORIA DA COMPUTAÇÃO

Uma Especificação Formal para a Programação Dinâmica
L. V. Toscani, P.A.S. Veloso 477

Resolucion y Tableaux Semanticos para la Logica Temporal: Una Comparacion
A. R. Cavalli 487

Conception et Mise en Oeuvre d'un Systeme de Preuve pour un sous ensemble de CCS Sans Transmission de Valeurs
A. Soriano, J. Voiron 500

traves da negociaçao entre os PTAS-APLICAÇAO da transferen

Um Esboço de uma Lógica de Resolução de Problemas
M.A. Lopes, B.M. Acioly 516

Un Lenguage Formal de Especificacion de Processos Comunicantes para El Analisis de Redes
Ph. Jorrand, J.M. Pereira 526

SESSÃO 13 - COMUNICAÇÕES

Suporte de Execução para a Linguagem LANAC - II, utilizando as facilidades do DOS-8000 do SISCO MB-8000
V.A.D. Oleinki, O.B. Coelho 539

Implementação da Aritmética de Intervalo com Exatidão Máxima
M.B. Correia, R.A. Faria, A.R.S. Serra Jr, E.N. Silva 542

Suporte Eletrônico para um Espectrofotômetro Infravermelho por transformada de Fourier Rápida
L.F. Costa, M. Siu Li 545

Um Método para Busca em Textos Usando Pré-processamento
M.G.V. Nunes 548

Isis, un Sistema de Informacion Espacial Inteligente (Intelligent Spacial Information System)
J. Bosaans 551

Modelagem de Diálogos Gráficos através de Diagramas de Transição de Estado
C.M.D. Sasso Freitas 555

Facilidades de Processamento Numérico e Gráfico em um Interpretador PROLOG
A.E.C. Pereira, M.C. Monard, R. Allan 560

Estágio atual do Desenvolvimento de um Interpretador PROLOG em Pascal Seqüencial
A.L.B. Dieterich 563

Um Modelo para a Análise de Cartas Sinóticas por Computador
S.R.M. Pellegrino, J.A.G. Pereira 567

Implementação da Linguagem Grafcet em um Controlador Programável
M. Roisemberg, L.A. Carvalho 570

SESSÃO 14 - BANCO DE DADOS

Uma Ferramenta para Projeto Modular de Banco de Dados
M.A. Casanova, L. Tucherman, A.L. Furtado 575

O Conceito de Tempo em SGBDs Generalizados
J. P. M. de Oliveira 586

Um Sistema de Almacenamiento y Recuperación de Informacion para Facilitar tanto Processos Pontuales como en Ordem Lexicográfico
F. Aurtenechea 596

Un Diccionario de Datos para un Sistema de Apoyo a La Toma de Decisiones
S. Abad 606

Projeto do Esquema Conceitual para Sistemas de Informação
C.L.G. Medeiros, U. Schiel 619

Computarizacion de Informacion Bibliografia, Aplicacion de Teoria de Automatas y de Consultas Interactivas
S.M. Tenzer, R. Lavega 635

SESSÃO 15 - SISTEMAS OPERACIONAIS

An Extension in the Definition of a Petri Execution
R. H. Gallard 647

SONIX - Uma Implementação do UNIX em Pascal Concorrente
C.M. da Costa, S.S. Toscani, T.B. Corso 655

S-SHELL - A Linguagem de Comandos do Sistema SONIX
C.M. da Costa, S.S. Toscani, T.B. Corso 668

SESSÃO 16 - IV CONCURSO DE TRABALHOS DE INICIAÇÃO CIENTÍFICA
TRABALHOS PREMIADOS:

Sistema de Aquisição e Exibição de Sinais Analógicos para Microcomputadores
G.M. Stephan e W. Lobianco Jr. 680

Sistema de Pulverização Catódica para Deposição de Filmes Finos de Dióxido de Silício
C.R. Peter 692

Interface para Ligação de Traçador Gráfico a Sistema de Computação
T.I. Nunes, C.T. Silveira, C.A. Prolo e A. Guignatti, R.G. Telichevesky...704

TUTORIAL

Computers and Programming Languages for AI
Ph. Treleaven, I.G. Lima 713

E.B.Lucena*, P.R.F.Cunha** e J.A.S.Monteiro***

SUMÁRIO

Este trabalho tem o objetivo de descrever o desenvolvimento de uma solução modular para a programação da Estação de Transporte da Rede CEPINNE, bem como o mapeamento desta modularização para o ambiente de programação sequencial disponível na nossa universidade. As alternativas de integração deste software ao ambiente hospedeiro, bem como a opção escolhida são também apresentadas.

ABSTRACT

This work has the objective of describing the development of a modular solution for the programming of the CEPINNE Network Transport Station and the mapping between this modularization and the programming environment available in our university. The alternatives how to integrate this software to the host computer and the implemented option are also presented.

* Bacharel em Ciência da Computação (UFPE,1984), Sistemas Operacionais, Redes de Computadores, Sistemas Distribuídos, REDIS/UFPE, Av. Prof. Luiz Freire s/n - 50.000 - Recife - PE - (081) 271.2510

** Engenheiro Eletricista (UFPE,1974), Mestre em Informática (UFPE,1977), Ph.D. em Computação (Waterloo, Canadá,1981). Professor Adjunto do Deptº de Informática da UFPE e membro do Grupo de Redes de Computadores e Sistemas Distribuídos da UFPE. Metodologias de Programação, Sistemas Distribuídos, Redes de Computadores.

*** Engenheiro Eletricista (UFPE,1979), Mestre em Engenharia Elétrica (USP,1982), Professor Assistente do Deptº de Informática UFPE, Redes de Computadores, Especificação e Validação de Protocolos, Sistemas Distribuídos.

1. INTRODUÇÃO

São várias as razões que nos levam a interligar os diversos sistemas computacionais existentes. Dentre elas podemos citar a necessidade de compartilharmos de recursos (informações, potencial computacional, dispositivos, etc.), a necessidade de desenvolvimento de recursos humanos especializados, a necessidade de um meio de comunicação mais poderoso, confiável e barato, etc.

Foi com estes e outros objetivos em mente que a SEI (Secretaria Especial de Informática) elaborou, em janeiro de 1981, o Projeto CEPINNE (Centro Piloto de Serviços Públicos de Teleinformática para Aplicações em Ciência e Tecnologia - Região Norte - Nordeste) [9], o qual envolveria a interligação dos diversos recursos computacionais disponíveis nas universidades das regiões Norte e Nordeste.

A participação da UFPE (sob o comando de seu Grupo de Redes de Computadores e Sistemas Distribuídos - REDIS/UFPE) neste projeto passou a ser mais efetiva a partir de janeiro de 1984, quando da celebração de um convênio entre UFPE, UFPB, EMBRATEL e FUNAPE (Fundação de Apoio a Pesquisa e Extensão da Universidade Federal da Paraíba), que visava ao desenvolvimento, implementação e testes de um Protocolo de Transportes para a Rede CEPINNE.

Este artigo tem o objetivo de descrever o desenvolvimento de uma solução modular para a programação da nossa Estação de Transporte, bem como o mapeamento deste modularização para o ambiente de programação, desprovido de paralelismo, disponível na UFPE. Na seção 2, descrevemos, sucintamente, o Protocolo de Transporte adotado. A seção 3 apresenta a estrutura modular do software (em termos de processos e monitores) da nossa Estação de Transporte e o mapeamento desta estrutura para o nosso ambiente. As alternativas de integração deste software ao ambiente hospedeiro, bem como a opção escolhida são apresentadas na seção 4. Finalmente, na seção 5 tecemos algumas considerações finais e identificamos futuros trabalhos que darão continuidade a este projeto.

2. PROTOCOLO DE TRANSPORTE ADOTADO PARA A REDE CEPINNE

Sabe-se que um dos fatores mais importantes no projeto de uma Rede de computadores é a forma com que os sistemas abertos comunicantes conversarão de modo a prover uma comunicação confiável, eficiente e sincronizada. A fim de reduzir a complexidade do projeto e tendo em mente a padronização da interconexão de sistemas abertos, a ISO ("International Organization for Standardization") elaborou o Modelo de Referência OSI (Open Systems Interconnection), o qual estrutura as redes em sete níveis ou camadas, com funções bem definidas, onde cada camada utiliza-se diretamente dos serviços fornecidos pela camada imediatamente inferior [11]. As regras e convenções segundo as quais uma entidade de nível N "conversa" com outra entidade de mesmo nível são globalmente conhecidas como Protocolo de Nível N.

Nesta arquitetura, o Protocolo de Transporte, o objeto de nosso trabalho, corres-

ponde ao protocolo de nível 4. O principal objetivo deste nível é prover uma transferência fim a fim confiável e eficiente. Tal transferência deverá envolver, eventualmente, o fornecimento de serviços tais como multiplexação de circuitos virtuais, quebra e remontagem de mensagens, controle de fluxo, controle de erros fim a fim, recuperação de falhas no nível de Rede (nível 3 na arquitetura do Modelo OSI) — RESET's e RESTART's — , etc. Podemos dizer, em outras palavras, que a camada de transporte procura preencher o eventual espaço existente entre o nível de qualidade de serviço requisitado pelo nível de Sessão (nível 5 na arquitetura do Modelo OSI) e os serviços efetivamente oferecidos pelo nível de Rede.

Quanto ao protocolo de Transporte para a Rede CEPINNE, estabeleceu-se, por questão de tempo, adotar como referência uma especificação preliminar de um protocolo de Transporte desenvolvido pela PUC/RJ [5]. Esta especificação foi revisada, por membros da UFPE, UFPB e PUC/RJ, quando da reunião do LARC (Laboratório Nacional de Redes de Computadores), ocorrida em abril de 1984, sendo efetuados alguns esclarecimentos, retificações e simplificações. A partir destas discussões foram elaborados dois documentos: a especificação dos serviços providos e requeridos por este protocolo de Transporte [1] e a especificação deste protocolo propriamente dito [8]. Decidiu-se também, com o intuito de agilizar a implementação, por não se implementar, ao menos inicialmente, serviços opcionais que o protocolo se propõe a prover, tais como multiplexação, recuperação de falhas, controle de erros fim a fim, transferência não orientada a conexão, delegando estas funções aos níveis superiores.

Este protocolo de Transporte é baseado na troca de fragmentos (unidades de informação de controle e trechos de mensagens), os quais são divididos em dois grupos: estabelecimento e encerramento de conexões (= ligações) e transferência de dados. São eles:

ESTABELECIMENTO E ENCERRAMENTO DE CONEXÕES

- PCONEX (Pedido de Conexão)
- CONACEITA (Aceitação de Conexão Solicitada)
- REJCON (Rejeição de Conexão)
- ACKAC (Confirmação de Aceitação de Conexão)
- FCONEX (Encerramento de Conexão)
- ACKFE (Confirmação de Encerramento de Conexão)

TRANSFERÊNCIA DE DADOS

- DADOS (Transporte de Dados Normais)
- INTERRUPÇÃO (Transporte de Interrupção)
- TELEGRAMA (Transporte de Telegramas)
- ACKDADOS (Confirmação de Recepção de Dados e/ou Interrupção)

As conexões deste protocolo permitem uma transferência full-duplex, ou melhor, o fluxo de dados pode ocorrer nos dois sentidos simultaneamente, e o encerramento da conexão

num sentido não implica necessariamente no encerramento no sentido oposto (a menos do aborto da conexão). As temporizações envolvidas neste protocolo são basicamente duas: a temporização de retransmissão de fragmentos e a temporização de inatividade remota (tempo máximo, determinado pelo usuário, que a Estação de Transporte suportará sem receber nenhum fragmento da entidade remota). O serviço requerido, por este protocolo, do nível inferior é o serviço de Circuitos Virtuais do nível 3 da Recomendação X.25 [3]. Maiores detalhes acerca deste protocolo e seus serviços podem ser encontrados nas referências [8] e [1], respectivamente.

3. ESTRUTURA MODULAR DO SOFTWARE DA ESTAÇÃO DE TRANSPORTE

Procuramos estruturar o software da Estação de Transporte em módulos de maneira que pudéssemos atacar o problema por etapas, o que consequentemente facilitaria a implementação e os testes desta Estação.

Num ambiente apropriado, seria possível também a exploração de todos os pontos de paralelismo existentes em um software desta natureza. Por exemplo, o tratamento da chegada de primitivas vindas do(s) nível(eis) superior(es) poderia, desde que haja algum mecanismo de sincronização do acesso às variáveis comuns, ser feito em paralelo com o tratamento da chegada de primitivas vindas do nível inferior e com o tratamento das indicações de estouro de temporizadores.

De um modo geral, esta estrutura derivou, em certos aspectos, da metodologia de programação proposta em [10]. Esta metodologia se baseia numa programação por eventos, sendo ocorrência destes eventos representada pela presença de dados. No nosso caso, a ocorrência de eventos é indicada pela presença de primitivas nas filas dos Monitores de Interface. Tais monitores podem ser controlados por um gerenciador de interfaces como em [10]. A ocorrência destes eventos (presença de primitivas na filas de recepção) irá disparar a execução de um certo conjunto de ações (consumo da indicação de ocorrência do evento, envio de novas primitivas, disparo de temporizações, atualização de variáveis, etc.).

A estrutura do software da Estação de Transporte é apresentada no diagrama da figura 3.1, no qual os arcos direcionados indicam o sentido do fluxo de dados, os módulos entre circuitos simbolizam Monitores (entidades passivas) e os módulos entre retângulos simbolizam Processos (entidades ativas). A comunicação entre cada um dos processos é feita com o auxílio de dois monitores de interface (que contêm as filas de recepção e transmissão de primitivas e podem ser controlados por um gerenciador de interfaces). A seguir apresentaremos a estrutura e, em seguida, descreveremos cada um dos seus módulos.

PROCESSOS

Pi:

Representam os diversos processos que usufruem dos serviços providos pela Estação de Transporte.

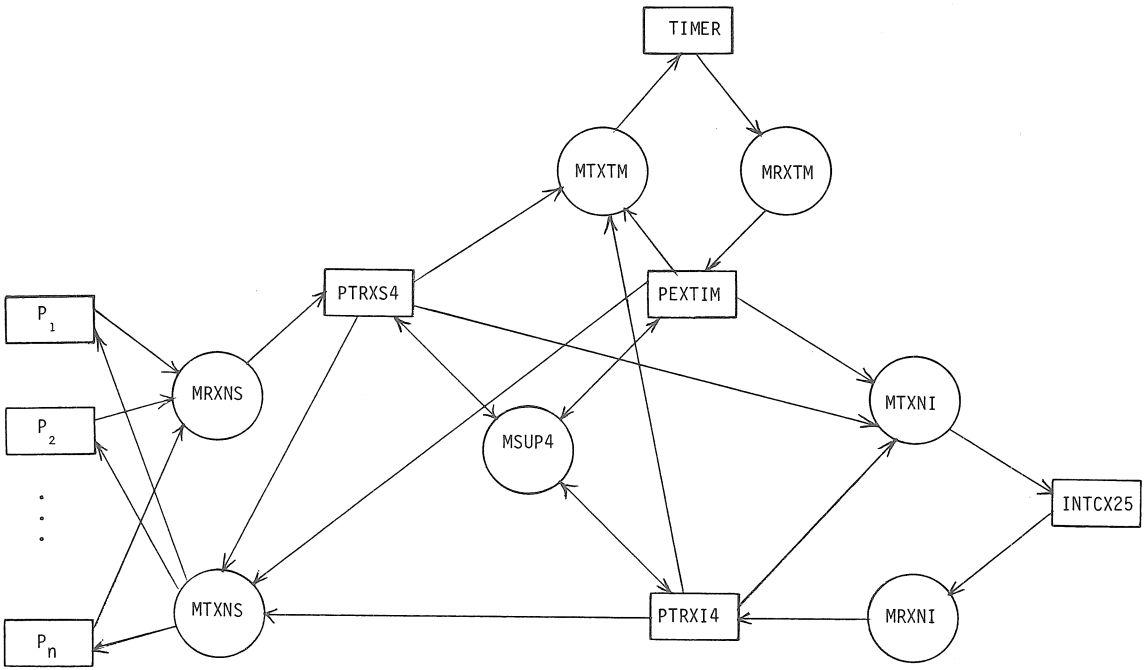


FIG. 3.1 - ESTRUTRURA DO SOFTWARE DA ESTAÇÃO DE TRANSPORTE

TIMER:

Processo responsável pelas temporizações do sistema. Não é propriamente um módulo da Estação de Transporte. Guarda as solicitações de temporizações (identificação do processo solicitante, identificação do tipo de temporização e conexão, tempo solicitado), computa este tempo solicitado e notifica o processo solicitante quando da ocorrência de time-out (através do envio de uma indicação sob a forma de primitiva). Pode ser utilizado simultaneamente pela Estação de Transporte e qualquer outro processo usuário que dele necessitar.

PEXTIM:

Trata da ocorrência do evento "estouro de temporizador" (evidenciado pela presença de uma indicação na fila de recepção) na Estação de Transporte. Dentre as ações contidas no conjunto de tratamento para este evento podem ser citadas o consumo da indicação, solicitações de novas temporizações, atualização do bloco de contexto da ligação, envio de primitivas aos níveis superior e/ou inferior, etc.

PTRXS4:

Processo responsável pela execução de ações pela ocorrência de eventos representados pela presença de primitivas vindas do(s) nível(eis) superior(es), tais como ABRECON, ESPERACON, FECHACON, ENVMSG, ENVINT, etc. Podemos citar ações como consumo da indicação, solicitação de portas, blocos de controle de ligação, envio de primitivas de solicitação de circuito virtual (ESTABCV), etc.

PTRXT4:

Processo responsável pela execução de ações em virtude da ocorrência de eventos representados pela presença de primitivas originadas do nível inferior (no nosso caso, a interface com o conversor CX.25). Tais ações incluem o consumo de indicação, o envio de primitivas que sinalizem ao nível superior e estabelecimento de conexões, a ocorrência de abortos de conexões, a chegada de mensagens, etc.

INTCX25:

Este processo efetua a interface entre a Estação de Transporte e o conversor CX.25.. Evidentemente não pode ser considerado efetivamente como um módulo da Estação de Transporte. Suas tarefas englobam o tratamento de primitivas vindas da Estação de Transporte (ex. ESTABCV, LIMPACV, TRPACOTE, etc.), o envio de primitivas de indicação ou resposta à Estação de Transporte (IND-RESET, RESP-ESTABCV, etc.), a interpretação e tratamento dos comandos vindos do conversor CX.25, etc. Maiores detalhes sobre esta interface podem ser obtidas na referência [6].

MONITORES

MRXNS:

Monitor de recepção de primitivas originadas do(s) nível(eis) superior(es)

(ABRECON, FECHACON, ABORTACON, etc.). Contêm a fila de recepção de primitivas do(s) nível(eis) superior(es). As operações permitidas são inserção de primitiva no final da fila e recuperação da primeira primitiva da fila.

MRXNI:

Monitor de recepção de primitivas originadas do nível inferior (INTCX25). Contêm a fila de recepção de primitivas enviadas pela interface com o conversor CX.25. (IND-ESTABCV, RES-TRPACOTE, etc.). As operações permitidas são as mesmas do MRRXNS.

MRXTM:

Monitor de recepção de indicações de time-out (primitiva ESTOURO-TEMPORIZADOR). Contêm a fila de notificações de time-out. As operações permitidas são as mesmas do MRXNS.

MTXNS:

Monitor de transmissão de primitivas destinadas ao(s) nível(eis) superior(es). Contêm a fila de transmissão de primitivas ao(s) nível(eis) superior(es) (CHEG-MENSAGEM, IND-ABORTO, RESP-ABRECON, etc.). As operações permitidas são as mesmas.

MTXNI:

Monitor de transmissão de primitivas ao nível inferior. Contêm a fila de transmissão de primitivas destinadas ao nível inferior (ESTABCV, TRPACOTE, LIMPACV, etc.). As operações permitidas são as mesmas.

MTXTM:

Monitor de transmissão de solicitações de temporizações ao Temporizador (TIMER). Contêm a fila de solicitações de temporizações (LIGA-TEMPORIZADOR). As operações permitidas são as mesmas.

MSUP4:

Monitor de supervisão da Estação de Transporte. Contêm as estruturas de controle da Estação de Transporte (Mapas de Portas, Mapas de Circuitos Virtuais, Blocos de Controle das Ligações, Blocos de Espera de Ligações, etc.). As operações permitidas são basicamente consulta e atualização.

3.1. MAPEAMENTO PARA O AMBIENTE COMPUTACIONAL DA UFPE

O sistema de computação hospedeiro disponível na UFPE é o DECsystem-10 (com Sistema Operacional em regime de Time-Sharing, versão 7.01A). O fato deste ambiente não nos permitir implementar facilmente regiões críticas controladas por mecanismos de sincronização tipo Monitores (uma vez que não dispomos no momento de nenhuma implementação de linguagem concorrente), praticamente inviabilizou a implementação da estrutura apresentada anteriormente. Desta forma, a maneira mais conveniente de permitir a comunicação entre os processos encontrada foi utilizar o software utilitário disponível no TOPS-10, IPCF ("Inter-Process Communication Facility") [4]. Com o uso deste utilitário, os processos se comunicam através da troca de mensagens ou pacotes IPCF. A cada processo estão asso-

ciados uma fila de transmissão e outra de recepção de pacotes. O gerenciamento destas filas é praticamente feito pelo TOPS-10. Os nossos monitores de transmissão e recepção (monitores de interface) foram mapeados para as filas de transmissão e recepção de pacotes IPCF. Os processos PTRXI4, PTRXS4 e PEXTIM que, na estrutura, concorriam ao acesso das mesmas informações (Bloco de Controle das Ligações, Mapas de portas, etc.) foram mapeados para um único processo. Evidentemente esta solução não é a mais eficiente e estruturada, entretanto foi a mais imediata que encontramos para o curto prazo de conclusão do projeto que dispúnhamos.

4. ALTERNATIVAS DE INTEGRAÇÃO E OPÇÃO ADOTADA

Existem basicamente três maneiras de se integrar a implementação de um protocolo no contexto de um Sistema Operacional hospedeiro [7]. A primeira é implementar o protocolo sob a forma de um ou mais processos usuários (abstração utilizada pela maioria dos Sistemas Operacionais a fim de prover um ambiente de execução para os programas dos usuários). A segunda forma é incorporar a implementação ao Kernel do Sistema Operacional. A última forma é implementar o protocolo fora da máquina hospedeira, em um processador de comunicação ou Front-end. A seguir idenficificaremos as vantagens, desvantagens e requisitos de cada opção.

A implementação do protocolo sob a forma de processo(s) usuário(os) é possivelmente a forma mais simples e a que exige menos alterações no ambinete hospedeiro, uma vez que não é necessário nenhuma modificação no Kernel do Sistema Operacional e são necessários apenas conhecimentos superficiais com respeito ao Kernel, haja vista que a implementação funcionará, em linhas gerais, como apenas mais um usuário do Sistema Operacional. Entretanto, esta opção é provavelmente a menos eficiente em termos de performance, pois o escalonamento de processos em um Sistema Operacional é uma fonte de atraso significativa.

Implementar o protocolo como parte do Kernel do Sistema Operacional possivelmente aliviará o overhead de escalonamento de processos e possivelmente facilitará o projeto de aplicações distribuídas, uma vez que poderíamos inclusive possibilitar a leitura e gravação de informações de/para a rede por processos usuários diretamente através de chamadas ao Sistema Operacional. Todavia esta alternativa possui várias desvantagens, entre as quais a necessidade de profundo conhecimento sobre o Sistema Operacional, o comprometimento da performance do Sistema Operacional como um todo e a dificuldade de manutenção do Kernel e da implementação do protocolo quando da ocorrência de mudanças (modificações de versão do Sistema Operacional, alterações no protocolo, entre outras).

A utilização de um processador de Front-end dedicado também possui suas vantagens e desvantagens. Entre as vantagens, isola o protocolo do Kernel do Sistema Operacional (consequentemente simplificando a implementação e manutenção do mesmo), é mais eficiente (uma vez que praticamente torna o hospedeiro livre de tarefa de comunicação), é mais versátil (haja vista que, com algumas modificações, pode ser utilizada por diferentes máqui

nas hospedeiras). Dentre as principais desvantagens, podemos citar o custo adicional do processador dedicado, a necessidade de uma comunicação, por mais simples que seja, entre a máquina hospedeira e o processador front-end, a dificuldade de interação com os níveis de protocolos adjacentes, etc.

Face ao exposto acima, resolvemos optar por uma alternativa que, poderíamos dizer, combina as características das duas primeiras alternativas. Em outras palavras, nossa implementação foi feita sob a forma de processos, mas que são inicializados juntamente com os processos hibernantes do Sistema Operacional TOPS-10, com certo nível alto de prioridade (mecanismo disponível no ambiente do TOPS-10). Dentre os fatores que nos levaram a tal opção, podemos citar a necessidade de termos, o mais rápido possível, instalada a nossa implementação da Estação de Transporte (o que descartaria a implementação num Front-end ou incorporação direta ao Kernel), a relativa simplicidade de implementação e manutenção, a necessidade de uma melhor interação entre as camadas adjacentes da arquitetura (uma vez que os serviços do nível de Rede já seriam fornecidos por um conversor X.25, o que já implicaria no desenvolvimento de uma interface) e a preferência por não provocar consequências (perda de performance do Sistema Operacional) em toda uma comunidade usuária (o que praticamente inviabilizaria a incorporação ao Kernel).

5. CONSIDERAÇÕES FINAIS E DIRECIONAMENTOS FUTUROS

Esta Estação de Transporte encontra-se atualmente implementada em Pascal seqüencial e testada localmente no DEC-10 da UFPE [2]. Estamos atualmente iniciando o desenvolvimento de uma metodologia de testes para protocolos de comunicação que poderá ajudar de uma maneira mais eficaz os testes integrados UFPE/UFPB do Nível de Transporte da Rede CEPINNE. Estamos na expectativa da instalação do nó REXPAC e no momento, do fornecimento dos conversores CX.25 às universidades envolvidas para que seja possível esta integração. Para dar continuação aos trabalhos relacionados com este projeto, prevemos a inclusão de novos serviços na nossa implementação, tais como multiplexação, recuperação de falhas, controle de erros fim a fim, etc. Também está prevista a especificação do protocolo e seus serviços utilizando-se técnicas mais formais que permitam uma validação mais efetiva deste protocolo.

AGRADECIMENTOS

Gostaríamos de agradecer o apoio financeiro concedido pela EMBRATEL e pela SEI, bem como o apoio técnico fornecido pelo Núcleo de Processamento de Dados da UFPE.

REFERÊNCIAS

[1] CUNHA,P.R.F.; MONTEIRO,J.A.S.; LUCENA,E.B. - "Especificação dos serviços do Protocolo de Transporte para a Rede CEPINNE"; Relatório Técnico submetido à EMBRATEL, dezembro/1984.

[2] CUNHA,P.R.F.; MONTEIRO,J.A.S.; LUCENA,E.B. - "Implementação de um Protocolo de Transporte para a Rede CEPINNE"; Anais do 3º Simpósio Brasileiro sobre Redes de

Computadores, Rio de Janeiro/RJ, abril/1985.

[3] CCITT - Comitê Consultatif Internacional Télégraphique et Téléphonique - "Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipament for Terminals Operating in the Packet Mode on Public Data Networks"; Recommendation X.25, Genebra, novembro/1980.

[4] DIGITAL EQUIPMENT CORPORATION - "Communication Between Processes: IPCF"; Software Netbook 4, julho/1982.

[5] MENASCÉ,D.A. et al. - Especificação Preliminar de um Protocolo de Transporte para Redes Públicas que usam Circuitos Virtuais"; Relatório Técnico, Deptº de Informática, PUC/RJ (Contrato C.GCD-004/80), dezembro/1981.

[6] MONTEIRO,J.A.S.; CUNHA,P.R.F.; RODRIGUES,J.S.F. - "Uso de um Conversor 'START/STOP' - X.25 para o Acesso de Computadores a REDES PÚBLICAS"; Anais do 3º Simpósio Brasileiro sobre Redes de Computadores, Rio de Janeiro/RJ, abril/1985.

[7] NBS - National Bureau of Standards - "Specification of a Transport Protocol for Computer Communications, Volume 5: Guidance for the Implementator"; Institute for Computer Sciences and Technology, U.S. Department of Commerce, janeiro/1983.

[8] SAUVÉ,J.P. et al. - "Especificação do Protocolo de Transporte para a Rede CEPINNE"; Relatório Técnico submetido à EMBRATEL, dezembro/1984.

[9] SEI - Secretaria Especial de Informática - "Projeto CEPINNE"; janeiro/1981.

[10] QUEIROZ,R.J.G.B. - "Uma Metodologia de Programação para Implementação de Protocolos"; Tese de Mestrado, Departamento de Informática, Universidade Federal de Pernambuco/PE, agosto/1984.

[11] ZIMMERMAN,H.; DAY,J.D. - "The OSI Reference Model"; Proceedings of the IEEE, volume 71, dezembro/1983.

SNPUC - UM SERVIDOR DE NOMES PARA A REDPUC*

JOSÉ LAEDIO MEDEIROS** E DANIEL ALBERTO MENASCÉ***

SUMÁRIO

O problema enfocado neste trabalho é a denominação e localização de um ambiente de Redes Locais. Um objeto é qualquer entidade (lógica ou física) do ambiente. Em particular, este artigo apresenta uma especificação de um Servidor de Nomes (SN), desenvolvido para a rede local REDPUC. As principais funções do SN apresentado são: o estabelecimento do mapeamento entre nomes de processos e "portas" do nível de transporte e gerenciamento do espaço de nomes de processos.

ABSTRACT

This work addresses the problem of naming and placement of object in a Local Area Network environment. An object is any logical or physical entity of the environment. In particular, this paper contains a specification of a Name Server (NS) developped for the local network REDPUC. The main functions of this name server are: dynamic mapping between process names and transport level ports, as well as the management of the process name space.

* Trabalho parcialmente financiado pelo Banco do Nordeste do Brasil, pela FINEP e pela EMBRATEL S.A. (contrato C.DCD-002/82)

** M.Sc. (PUC/RJ, 1985); Banco do Nordeste do Brasil, Fortaleza - Ceará.

*** Professor Associado, Depto. de Informática, PUC/RJ; Rua Marques de São Vicente. , 225, CEP - 22453 - Rio de Janeiro - RJ

1. INTRODUÇÃO

Este trabalho enquadra-se no contexto de computação distribuída. O problema enfocado é a denominação e localização de objetos em um ambiente de Redes Locais. Um objeto é qualquer entidade (lógica ou física) do ambiente. Objetos podem ser individuais , tais como terminais, servidor de arquivos, pessoas, processos ou grupos de outros objetos, como por exemplo: uma lista de controle de acesso de usuários [9]. A seção dois apresenta conceitos básicos sobre servidores de nomes e descreve um sistema relacionado com o aqui descrito - o "Clearinghouse" [9]. A seção três descreve o servidor de nomes SNPUC. Nesta seção são apresentadas as primitivas oferecidas pelo SNPUC, uma descrição informal do seu funcionamento bem como o mecanismo de eleição utilizado para tornar a operação do SNPUC robusto em caso de falhas. O artigo conclui com algumas considerações finais sobre o ambiente de implementação.

2. CARACTERIZAÇÃO DE UM SERVIDOR DE NOMES

2.1 - Conceituação de Servidor de Nomes

A nomeação e localização de objetos em ambientes distribuídos deixa de ser trivial na medida em que esses ambientes tornam-se amplos, o que dificulta a identificação dos objetos e acesso a esses. Há portanto, a necessidade da introdução de um espaço de nomes que seja global à rede, ou interrede (no caso de interconexão de redes), e algo que transforme este espaço de nomes em endereços.

De uma maneira restrita, um Servidor de Nomes (SN) seria um agente capaz de oferecer uma forma padronizada de atribuição de nomes a objetos, bem como permitir que esses objetos sejam referenciados simplesmente pelos seus nomes, independente de suas localizações (endereços) no ambiente distribuído. De outra forma este conceito pode ser estendido no sentido de atribuir ao SN os serviços de mapear um espaço de nomes em um espaço de propriedades, dentre as quais uma seria o endereço. Ou seja, cada objeto poderia ter associado a seu nome, um conjunto de propriedades. Propriedades, estas caracterizadas como atributos referentes aos nomes.

Para efeito desse trabalho, definimos o SN como um agente responsável pelo fornecimento de primitivas que permitem, além da nomeação padronizada de nomes na interrede, e de sua principal função que é a associação dinâmica entre nomes de processos e "portas" no ambiente distirbuido, a administração tanto do espaço de nomes como do espaço de endereços, com o uso de um Banco de Dados distribuido.

Na seção seguinte, está descrito o "Clearinghouse" - um servidor de nomes desenvolvido pela Xerox [9].

O "Clearinghouse" é um agente descentralizado que tem como funções básicas nomear e localizar objetos em um ambiente distribuido. Utiliza-se de um Banco de Dados distribuido e replicado em alguns locais da interrede. A descentralização e réplica de informações, aumentam a eficiência quando de acessos a objetos, e a robustez do sistema no caso de falhas. Além dessas funções básicas, oferece facilidades para a administração do Banco de Dados e mecanismo de controle de acesso.

2.2.1 - Convenções de Nomes

Qualquer tipo de objeto é nomeado obedecendo a uma mesma convenção de nomes. Um nome é uma cadeia, não nula, de caracteres da forma <sub-cadeia1>@<sub-cadeia2>@<sub-cadeia3>, onde sub-cadeia1 é o nome local (L), sub-cadeia2 é o domínio (D) ao qual pertence o objeto denominado, e sub-cadeia3 identifica a organização (O). Desse modo temos L@D@O. Nenhuma das sub-cadeias pode conter ocorrências do caracter "@". A forma do nome com três cadeias, caracteriza uma hierarquia de três níveis, entretanto esta divisão é apenas lógica, pois os nomes são tratados com uma única cadeia. Os nomes são absolutos, ou seja, não existem dois objetos com o mesmo nome na interrede. Por exemplo, os nomes de objetos da rede PUC (organização), departamento de informática (domínio), seriam da forma <qualquer coisa>@info@puc. Um objeto, além do seu nome absoluto, pode ter um ou mais "aliases". "Aliases" são também absolutos. Um nome, ou é um nome distinto ou é um "alias", mas nunca ambos.

2.2.2 - Mapeamento

O "Clearinghouse" mapeia cada nome em um conjunto de propriedades associadas ao objeto que possui aquele nome. Uma propriedade é uma tupla ordenada, consistindo do nome da propriedade (nome-prop), um tipo de propriedade (tipo-prop) e o valor da propriedade (valor-prop). O "Clearinghouse" mantém mapeamentos da forma: {<nome-prop, tipo-prop1, valor-prop1>,...<nome-propk, tipo-propk, valor-propk>}, onde "k" pode ser qualquer.

Um "nome-prop" identifica uma propriedade especifica associada a um dado nome. Um "nome-prop" pertencente ao conjunto de propriedades associadas a um objeto é único no conjunto, isto é, nome-prop(i) $\neq$ nome-prop(j), se i $\neq$ j.

Um "valor-prop" é o valor atribuido a uma determinada propriedade. Há dois tipos de "tipo-prop". Tipo individual ou $\emptyset$, nesse caso "valor-prop" é tratado como uma cadeia de "bits" sem qualquer significação; e o tipo grupo ou 1, onde o "valor-prop" é tratado como um conjunto de nomes. Essa interpretação dada pelo "Clearinghouse" se refere à propriedade do nome, e não ao objeto que a possue. A seguir são mostrados alguns e

xemplos de mapeamento:

. tipo individual
Laedio@info@puc --> {<"usuário", Ø, qualquer coisa>,
<"senha", Ø, de autenticação>,
<"arquivos", 1, conjunto de arquivos do usuário>}

. tipo grupo
Alunos@info@puc --> {<"lista", 1,{"laedio@info@puc",
"nelson@info@puc",
... "ricardo"@info@puc}>}

O mapeamento de um nome em um conjunto de propriedade permite que diferentes usuários atuem sobre o mesmo "nome" simultaneamente, porém em propriedades diferentes. Da mesma forma, as propriedades do tipo grupo, que são associadas a um conjunto de nomes em "valor-prop", permite que vários usuários atuem sobre a mesma propriedade do mesmo "nome", em valores de propriedade distintos.

2.2.3 - Primitivas principais oferecidas aos usuários

2.2.3.1 - Sobre "nomes" e "aliases"

. adição de "nomes" ou "aliases";
. retirada de "nomes" ou "aliases";
. verificação da existência de "nomes" ou "aliases";
. troca de "nomes".

2.2.3.2 - Relativas a propriedades dos nomes

. verificação da existência de "propriedades" de "nomes";
. adição de "propriedades" a "nomes";
. retirada de "propriedades" de "nomes";
. troca "propriedades" de "nomes".

2.2.3.3 - Relativas a elementos de grupos de uma propriedade

. verificação da existência de "elementos" em grupos de propriedades;
. adição de "elementos" a grupos de propriedades;
. retirada de "elementos" de grupos de propriedades.

2.2.3.4 - Outras primitivas

. verificação genérica de "nomes";
. enumeração dos objetos de um domínio dado;
. enumeração dos domínios de uma organização dada;
. enumeração das organizações existentes na interrede;
. enumeração das propriedades de um "nome" dado.

2.2.4 - Estrutura do "Clearinghouse":

O "Clearinghouse" é composto pelo conjunto dos servidores distribuídos na interrede, os quais executam os serviços oferecidos aos usuários, ou pertinentes às tarefas in

rentes ao próprio "Clearinghouse". Além disso, oferece um conjunto básico de operações, que devem ser incorporados ao "software" do usuário, a fim de padronizar a comunicação entre o usuário e o "Clearinghouse".

Correspondendo a cada domínio (D), em cada organização (O), existe um ou mais servidores, cada um contendo uma cópia de todos os mapeamentos da forma <qualquer nome>@D@O. Estes servidores são chamados de Servidor Domínio (SD). Há pelo menos um SD para cada domínio no ambiente distribuido. Cada SD em uma organização (O), tem o nome da forma <qualquer nome>@O@servidor, mapeando o endereço de rede do servidor : <qualquer nome>@O@servidor --> {... <"endereço", Ø, endereço de rede> ...}.

Os mapeamentos de todos os SD's (D), de uma organização (O) (<qualquer nome>@O@servidor), estão contidos na porção do BD mantido por um servidor chamado de Servidor Organização (SO). Há no mínimo um servidor deste tipo para cada organização (O). Cada SO tem o nome da forma <qualquer nome>@servidor@servidor mapeando-o no seu endereço de rede: <qualquer nome>@servidor@servidor --> {...<"endereço", Ø, endereço de rede>...}.

2.2.4.1 - Interconexão entre os componentes do "Clearinghouse"

O conjunto de primitivas incorporadas ao "software" do usuário, contêm o endereço de no mínimo um servidor "Clearinghouse", ou sempre pode localizar um localmente ou em outro local da interrede, através de uma mensagem do tipo difusão.

Cada servidor "Clearinghouse" (SC) conhece o nome e endereço de cada SC na interrede. Desse modo, qualquer SC pode comunicar-se com qualquer SO da interrede diretamente.

Em qualquer solicitação do usuário ao "Clearinghouse", mesmo o pior caso, o número de SC's contactados não excede a três: servidor cujo endereço é conhecido, um SO que contêm a organização do "nome", e um SD dentro da organização.

Esta estrutura permite o uso de um algoritmo relativamente simples para o gerenciamento do "Clearinghouse". Entretanto requer que cópias de mapeamentos de todos os "nomes" da forma <qualquer coisa>@servidor@servidor (domínio organização) sejam armazenados em todos os servidores.

2.2.5 - Administração do "Clearinghouse"

As tarefas administrativas do "Clearinghouse" incluem os serviços de gerenciamento do espaço de nomes e de nomes de propriedades; inclusão de novas redes; decisão sobre como distribuir uma organização e objetos dentro dos domínios; adição, troca e retirada de serviços (tais como serviços de correio, servidores de arquivos, e os serviços próprios do "Clearinghouse"); adição e retirada de usuários; manutenção de

"senhas" dos usuários, etc); e a manutenção de uma lista de controle de acesso e outras características de segurança da rede.

A seguir estão descritas, sucintamente, algumas das tarefas do "Clearinghouse". É assumido que o "Clearinghouse" já está instalado na interrede.

2.2.5.1 - Gerenciamento do Espaço de Nomes

A alocação de nomes é gerenciada por uma autoridade de nomeação, a qual é responsável por garantir a existência de nomes absolutos. Há autoridades de nomeação para nomes de objetos, nomes de domínio, nomes de organização e nomes de propriedades.

2.2.5.2 - Adição de um novo usuário

O administrador do sistema, primeiro tenta incluir o novo usuário, através da chamada da rotina ADICIONA NOME (se não conseguir devido ao "nome" já existir, possibilita ao usuário substituir o "nome" dado). Em seguida, de acordo com as informações prestadas pelo novo usuário, registra sua "senha", seus privilégios, etc.

2.2.5.3 - Adição de um novo servidor (do usuário)

O administrador do sistema solicita ao componente do "Clearinghouse" existente no "software" do usuário, que este registre o "servidor" no SD apropriado. O registro , envolve no mínimo o nome do servidor, sua "senha" e seu endereço de rede. (São usadas para isso as funções ADICIONA NOME e ADICIONA NOME de PROPRIEDADE).

2.2.5.4 - Adição de novo servidor do "Clearinghouse"

O servidor a ser adicionado se enquadra em um dos seguintes casos: 1) é um SD para um novo domínio; 2) é um SD para um domínio existente; e 3) é um SD para uma nova organização. Assumindo que não existe conflito na nomeação, e que o " solicitante" tem permissão para requisitar tal serviço, teriamos de forma simplista os seguintes passos: 1) é atribuido um nome ao novo servidor; 2) se o novo servidor é de uma nova organização, o nome da nova organização é registrado, através da autoridade central de nomeação; 3) se o servidor é um novo domínio, o nome do domínio é registrado adequadamente; 4) o administrador decide que porção do BD do "Clearinghouse deve o novo servidor manter; 5) registra os dados que compõem o novo domínio (nomes, endereços, "senhas", etc. através das primitivas tais como ADICIONA NOME, ADICIONA PROPRIEDADES de NOMES, etc.); 6) solicita a todos os SC's, a inclusão do novo "servidor".

O "Clearinghouse" pode ser visto com mais detalhes na referência [9].

3.1 - Descrição do Servidor de Nomes (SNPUC)

O SNPUC situa-se no nível imediatamente acima do Protocolo de Transporte (nível 4)

O SNPUC é um agente descentralizado que tem como função básica localizar "processos", através de seus nomes no ambiente distribuido, e transformar esse espaço de nomes em um espaço de endereços. O espaço de nomes é composto por todos os nomes de processos conhecidos na interrede (que estão habilitados a uma conexão do nível de transporte ou já estão ativos em conexões já estabelecidas). Dessa forma as solicitações dos usuários não se restringem a informações disponíveis localmente, e sim, são estendidas ao ambiente das redes interligadas. Independente de sua localização, o mapeamento de nomes em endereços é transparente para os solicitantes dos serviços oferecidos pelo SNPUC.

O SNPUC considera objetos (nomes de processos) pertencentes a dois tipos de domínios:

tipo usuário (rede) - que engloba os processos pertencentes às redes locais

tipo servidor - que compõem os processos executores das funções do SNPUC.

3.1.1 - Convenções de Nomes

Todos os processos são nomeados obedecendo a uma única convenção. Um nome é uma cadeia, não nula, de caracteres da forma: sub-cadeia1/sub-cadeia2, onde sub-cadeia1 identifica o domínio (rede) ao qual pertence o processo denominado e sub-cadeia2 é um nome local (dentro da rede).

Os nomes são absolutos, ou seja, não existem dois processos com o mesmo nome dentro da interrede. Por exemplo, os nomes de processos da REDPUC são da forma: PUC/qualquer-coisa.

Nenhuma das duas sub-cadeias pode conter ocorrências do caracter "/".

A forma do nome com duas sub-cadeias caracteriza uma hierarquia de dois níveis, entretanto esta divisão é apenas lógica pois os nomes são tratados como uma única cadeia.

Para nomeação de servidores e interfaces de servidores é utilizada a mesma filosofia explicitada acima, sendo observado o seguinte: sub-cadeia1 = identificação de redes e sub-cadeia2 = identificação do servidor. Por exemplo, o servidor de nomes local da REDPUC é da forma: PUC/SNL.

3.1.2 - Descrição da arquitetura do SNPUC

O SNPUC é composto pelo conjunto dos Servidores de Nomes Locais - SNL's (um para cada rede), e de todas as Interfaces dos SNL's - ISNL's (uma para cada estação de cada rede).

Cada SNL tem acesso ao Banco de Dados da Rede (BDR) que contém:

i) mapeamento para o subconjunto de nomes que foram o domínio de rede local:

Ri/qq.nome -> { endereço do processo na rede local }

ii) mapeamento para o domínio servidores da interrede e interfaces da rede local:

Ri/Si -> { endereço do servidor na interrede }
Ri/ISi -> { endereço da interface na rede }

Cada ISNL tem acesso ao Banco de Dados da Estação (BDE) que contém:

i) mapeamento para o subconjunto dos usuários da estação do domínio rede local:

Ri/qq.nome -> { endereço da estação }

ii) mapeamento para o domínio Servidores do SNPUC e interfaces da rede local:

Ri/Si -> { endereço do servidor na interrede }
Ri/ISi -> { endereço da interface na rede }

A seguir (na fig. 3.1) é mostrada graficamente a arquitetura do SNPUC.

3.1.3 - Serviços Oferecidos pelo SNPUC aos usuários

1. ADIciona NOME	(ADINOME)
2. RETira NOME	(RETNOME)
3. MUDa NOME	(MUDNOME)
4. VERifica NOME	(VERNOME)
5. ENUmera NOMEs	(ENUNOME)
6. MUDa ENDEreço do nome	(MUDENDE)
7. CONEXAO dinâmica entre nomes e portas	(CONEXAO)
8. ENUmera SERVidores	(ENUSERV)

3.1.3.1 - Especificação das Primitivas

1. ADINOME
ENTRADAS : nome (de acordo com a convenção de nomes)
SAIDAS : cret (código de retorno)
DESCRIÇÃO : Caso ainda não exista o nome na rede local, referido nome (se BDR e BDE tiver espaço) é adicionado ao domínio de rede correspondente, tanto no BDR como no BDE (da estação do processo solicitante), com mapeamento para o endereço de origem do pedido (endereço do ISNL de origem)
Ri/nome dado -> { endereço do ISNL de origem }.

CÓDIGOS DE RETORNO :
Ø - INSERÇÃO OK;
1 - "nome" JÁ EXISTE;
2 - BD SEM ESPAÇO;
3 - REDE DO "nome" NÃO É LOCAL.

2. RETNOME
ENTRADAS : nome
SAIDAS : cret (código de retorno)
DESCRIÇÃO : caso "nome" dado exista na estação local, ele é retirado dos BDR e BDE nos quais está armazenado.

CÓDIGO DE RETORNO :
Ø - RETIRADA OK;
1 - "nome" NÃO EXISTE.

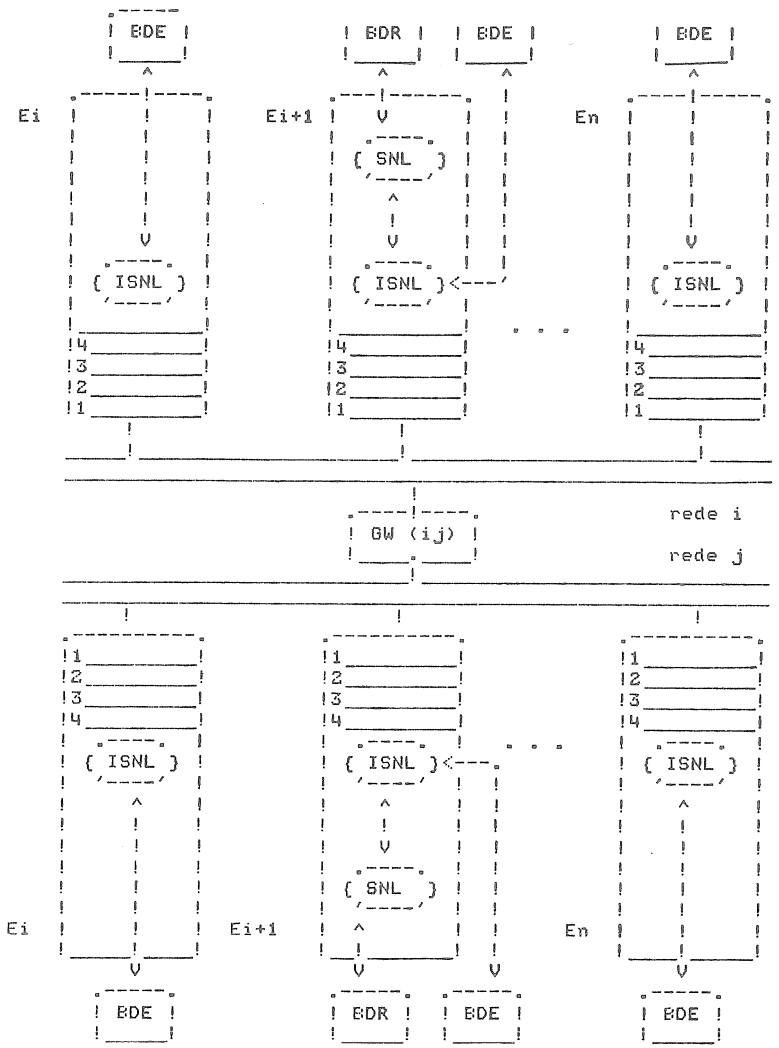


fig. 3.1 - arquitetura do SNPUC na interrede

LEGENDA: BDE- banco de dados da estação
BDR- banco de dados da rede (cada
SNL- servidor de nomes local (cada rede)
ISNL- interface do SNL (cada estação)
Ei - estação(i)
GW(i,j) - gateway de ligação da rede(i) com a rede(j)
1 - nível físico
2 - nível de acesso a barra
3 - nível interredes
4 - nível de transporte

3. MUDNOME

ENTRADAS : nome-novo
nome-velho

SAÍDAS : cret (código de retorno)

DESCRIÇÃO : caso "nome-velho exista na estação de origem da solicitação, "nome-velho e ainda não exista, a troca é realizada (no BDR e BDE)

CÓDIGO DE RETORNO : Ø - TROCA DE "nome" OK;
1 - "nome-novo" NÃO ACEITO;
2 - "nome-velho" NÃO EXISTE;
3 - "nome-novo" JÁ EXISTE.

4. VERNOME

ENTRADAS : nome

SAÍDAS : ender-nome (rede, módulo)
cret (código de retorno)

DESCRIÇÃO : caso "nome" exista na interrede (no BDE, ou no BDR local, ou no BDR de outra rede) é retornado o endereço correspondente, no qual o "nome" está mapeado (rede, módulo).

CÓDIGO DE RETORNO : Ø - "nome" EXISTE;
1 - "nome" NÃO EXISTE.

5. ENUNOME

ENTRADAS : tipo-enumeração
1 - para enumeração de todos os nomes cadastrados no módulo de origem do pedido;
2 - para enumeração de todos os nomes cadastrados na rede local.

SAÍDAS : nome(i), ender-nome(i) (relação dos nomes e respectivos endereços de nome(n), ender-nome(n) acordo com o tipo 1 ou 2)
...

DESCRIÇÃO : caso tipo-enumeração seja:
1 - são retornados ao processo solicitante todos os nomes e respectivos endereços (do domínio usuário) cadastrados no BDE local da ISNL de origem;
2 - são retornados ao processo solicitante todos os nomes e respectivos endereços (do domínio usuário) cadastrados no BDR da rede local.

6. MUDENDE

ENTRADAS : nome (a ser transporado)
ender-nome (novo módulo do nome)

SAÍDAS : cret (código de retorno)

DESCRIÇÃO : caso "nome" exista na rede local e "novo-módulo" exista e esteja ativo, o mapeamento no BDR é alterado para o novo endereço, sendo o "nome" retirado do BDE do módulo de origem e adicionado no BDE no novo módulo.

CÓDIGO DE RETRONO : Ø - MUDANÇA DE ENDEREÇO OK;
1 - "nome" NÃO EXISTE;
2 - MÓDULO DO "ender-nome" NÃO EXISTE ou INATIVO;
3 - BDE DO "novo-módulo" SEM ESPAÇO.

7. CONEXÃO

ENTRADAS nome1 (nome do processo que solicita a conexão)
nome2 (nome do processo com o qual o nome1 deseja conectar-se)

SAÍDAS portal (porta local do processo solicitante)

porta2 (porta local do outro processo)
cret (código de retorno)

DESCRIÇÃO : caso exista porta disponível tanto no módulo ISNL de origem, como na estação da interface onde "nome" está armazenado (se "nome2" existir) e "nome1" já esteja cadastrado no BDE local e BDR da rede, ou possa ser cadastrado com sucesso nos BD's, são alocadas duas portas, uma no módulo local da ISNL de origem (associada ao processo de "nome1"), e outra na estação do BDE do "nome" (associada ao processo de "nome2"). Os números das portas alocadas são retornadas ao processo solicitante.

CÓDIGO DE RETORNO : 0 - "porta1" e "porta2" ALOCADAS OK;
1 - NÃO EXISTE "PORTA" DISPONÍVEL;
2 - "nome1" NÃO PODE SER CADASTRADO;
3 - "nome2" NÃO EXISTE NA INTERREDE;
4 - MÓDULO DO "nome2" NÃO EXISTE ou NÃO ESTA ATIVO.

8. ENUSERV

ENTRADAS : nenhuma
: nome(i), ender(i) (relação dos servidores e respectivos endereços
... na interrede)
nome(n), ender(n)

DESCRIÇÃO : É feita uma solicitação ao BDE da ISNL ao qual foi feito o pedido e devolvidos todos os nomes e endereços de servidores cadastrados no domínio servi - dor.

3.1.4 - Serviços necessários para funcionamento do BDR de cada Rede Local

1 - CRIação do BDR (CRIABDR)
2 - RECuperação do BDR (RECBDR)

3.1.4.1 - Especificação das Primitivas

1. Criação do BDR

ENTRADAS : nenhuma
SAÍDAS : cret (código de retorno)
DESCRIÇÃO : Caso ainda não exista o arquivo (ou arquivos) que compõem o BDR da rede local no disco onde o SNL foi ativado e haja espaço em disco, o SNL cria um BDR local (BDRrede.SNL) com o domínio servidor (servidores da interrede), envia o domínio servidor às ISNL's da rede local e solicita aos SNL's das redes interligadas a inclusão do servidor atiado;
Caso já exista o arquivo BDrede.SNL na estação do SNL, o SNL considera o arquivo como válido, atualiza o domínio servidor (servidores da interrede) e complementa seus serviços da mesma forma do primeiro caso.

CÓDIGO DE RETORNO : 0 - "BDR" CRIADO COM SUCESSO;
1 - DISCO SEM ESPAÇO PARA O "BDR".

2. Recuperação do Servidor

ENTRADAS : nenhuma
SAÍDAS : cret (código de retorno)
DESCRIÇÃO : Caso haja espaço no disco onde o SNL foi ativado, o SNL cria um BDR (BDRrede.SNL), recupera o domínio servidor (servidores da interrede) e domínio usuário (processo da rede local), informa aos SNL's das redes interligadas o novo endereço do SNL recuperado.

CÓDIGO DE RETORNO : 0 - "BDR" RECUPERADO COM SUCESSO;

1 - DISCO SEM ESPEAÇO RARA O "BDR".

3.2. Descrição Informal do Funcionamento do SNPUC

Para que os nomes de processos constem do espaço de nomes do SNPUC (pertençam a um BDR e a um BDE) existem apenas duas formas:

1) através da solicitação explícita de inclusão de nomes de processos nos BD's do SNPUC com o uso da primitiva ADINOME;
2) através da utilização da primitiva CONEXÃO (usada para a associação dinâmica entre nomes de processos e portas, a qual possibilita a comunicação entre dois processos na interrede), nesse caso o processo solicitante do serviço, se ainda não constar do espaço de nomes do SNPUC é devidamente cadastrado pela SNL de origem.

Todas as solicitações ao SNPUC (interrede) devem ser endereçadas à ISNL da estação de origem do usuário (endereço bem conhecido), que devolve: ou o resultado do pedido ou o fracasso da operação solicitada.

As interfaces (ISNL's) são ativadas quando da entrada em operação das estações de cada rede local, começando assim os serviços do SNPUC.

A primeira tarefa de cada ISNL, após criar ou ativar o BDE, é o envio de uma mensagem (do tipo difusão) para o SNL da rede local, solicitando-lhe seu endereço. Caso o SNL esteja ativo e devolva seu endereço, a ISNL solicitante insere, no BDE local, o endereço do SNL ativo e considera-se apta a oferecer as primitivas do SNPUC. Caso não receba nenhuma resposta em um tempo "T" segundos (TIMEOUT), assume que o SNL local ainda não está ativo e aciona o algoritmo de eleição da estação que deve ativar o SNL da rede (vide 3.3.1). Caso a estação seja a sua, a interface deve ativar o SNL solicitando-lhe que crie o BDR (CRIABDR). Do mesmo modo procede quando é reativada devido à recuperação de falha de "hardware" do módulo onde ela estava em operação.

Caso o módulo onde o SNL de uma determinada rede pare de funcionar, o SNL deve ser recuperado em outra estação da rede, sendo usado para tal o mesmo algoritmo de eleição (vide 3.2.1). Nesse caso, a ISNL do módulo eleito ativa o SNL e solicita-lhe a recuperação do BDR (RECBDR).

Para a execução das primitivas oferecidas aos usuários e dos serviços inerentes ao protocolo do SNPUC é usada de uma forma geral o esquema de comunicação entre os componentes do SN, mostrado na fig. 3.2.

AÇÕES DE CADA ENTIDADE :

1. USUÁRIOS
   . Solicitam a execução de primitivas à ISNL da estação local;
   . Recebem respostas das solicitações.

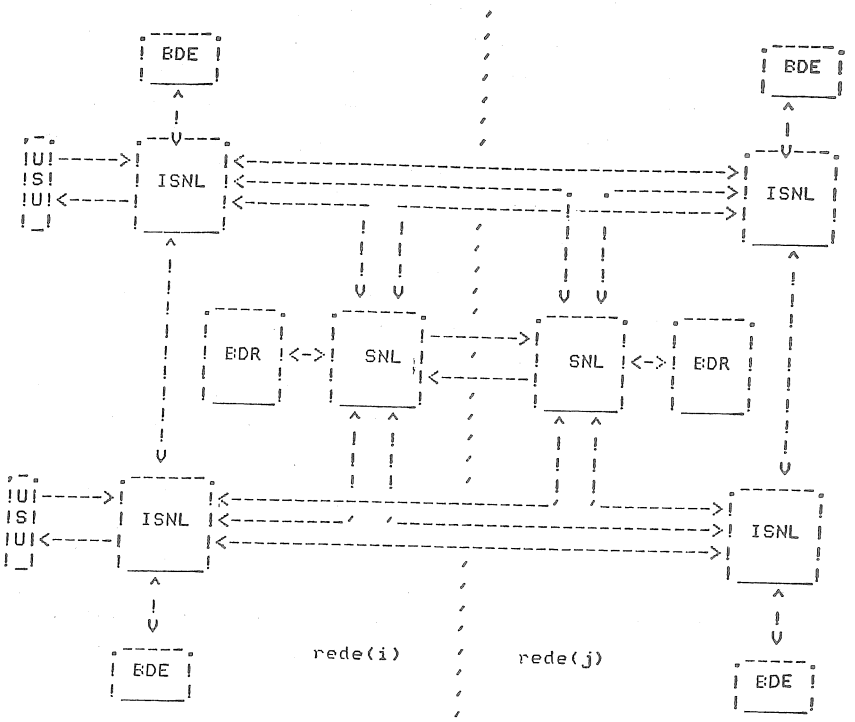


fig. 3.2 - esquema geral de comunicação do SNPUC

2. ISNL's
   . Recebem solicitações, ou dos usuários ou de outro componente do SNPUCC;
   . Consultam o BDE e executam um dos passos seguintes:
   1) caso atenda à solicitação localmente, ou detete qualquer situação de insucesso, devolve o solicitante a resposta adequada; fim;
   2) caso contrário, solicita ao SNL (da rede local ou de outra rede) o serviço; recebe a resposta da sua solicitação, ou do SNL ao qual enviou a requisição ou da ISNL do módulo endereçado pelo SNL requisitado; devolve a resposta adequada ao solicitante; fim;

3. SNL's
   . Recebem solicitações das ISNL's locais ou das SNL's (outra rede) e dependendo da solicitação, executa um dos passos seguintes:
   1) consulta o BDR (local) e devolve resposta ao INSL/SNL; fim.
   2) consulta o BDR e envia solicitação ao ISNL (de destino da rede local); fim.
   3) consulta o BDR e envia solicitação ao SNL de outra rede; fim.

A fim de um melhor entendimento, suponhamos que o processo "X", do módoulo "Mi" da rede "I", deseja se comunicar com o processo "Y", do módulo "Mj" da rede "J", e que o processo "X" já está cadastrado no BDE e BDR da rede "I" e o processo "Y" já se encontra no BDE de "Mj" e BDR da rede "j", além de existirem portas disponíveis em ambos os módulos, P1 e P2 respectivamente. O processo "X" se utiliza da primitiva "CONEXAO" passando os parâmetros <nome1> = X, <nome2> = Y e recebe de volta <cret = Ok> , <portal> = P1, e <porta2> = P2. Teríamos então o quadro mostrado na fig. 3.3

Na execução de suas primitivas o SNPUC utiliza-se do protocolo interredes (PI) e as ações realizadas são as seguintes:

1 - a interface do módulo de origem do processo "X" ao receber a solicitação CONEXAO (X,Y), verifica (através do PT) se existe porta disponível (hipótese=existe), consulta o BDE para ver se "X" já é cadastrado (hipótese = é cadastrado) ; em seguida envia uma mensagem ao SNL da rede "J" (endereço bem conhecido) <IP , os, id-ISNL, Y>;

2 - o SNL da rede "J" verifica-se "Y" existe na rede local (hipótese = existe) e envia ao ISNL do módulo onde "Y" consta do BDE local, uma mensagem solicitando alocação de porta para o processo "Y" <IP, os, id-ISNL, Y >;

3 - a ISNL (módulo de endereço de Y), verifica (através do PT) se existe porta disponível (hipótese = existe) e devolve a ISNL de origem de "X" a mensagem <IP,os,P2> informando a porta alocada, no caso P2;

4 - a ISNL de origem ao receber a mensagem da ISNL da outra rede, devolve ao usuário <OK,P1,P2> significando que houve sucesso na execução da primitiva e informando

quais portas foram alocadas.

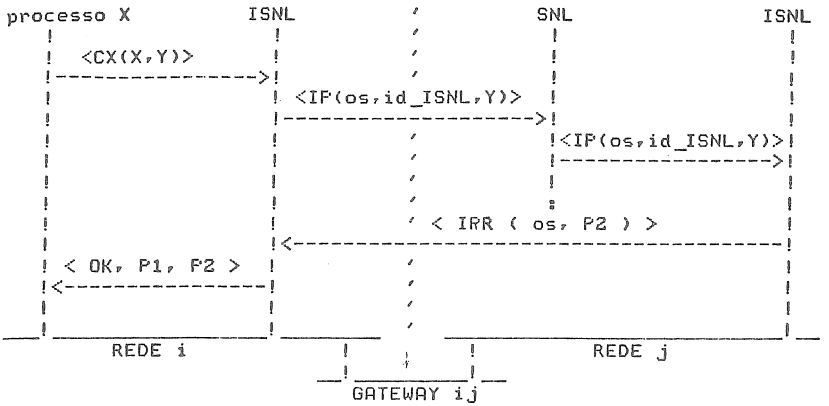


fig. 3.3. fluxo de mensagens trocadas na execução da primitiva CONEXAO

Legenda :

CX - CONEXÃO
IP - identificação da primitiva
IPR - identificação da primitiva resposta
os - ordem de solicitação
id-ISNL - identificação da ISNL requisitante
P1 - número da porta 1
P2 - número da porta 2

3.2.1 - Descrição informal do Algoritmo de Eleição (AE) da estação que deve ativar/recuperar o SNL em cada Rede Local

Os módulos elegíveis são aqueles ativos da Rede Local. São assumidas as seguintes hipóteses:

1. todas as estações executam o mesmo AE;
2. todas as estações tem implementados todos os níveis de comunicação: nível físico, nível de acesso a barra, protocolo interredes e protocolo de transportes;
3. nas transmissões de mensagens é usado o serviço de conexões confiáveis e com controle de erros fim-a-fim do PT [4].

Cada ISNL deve manter um vetor de estados que espelhe seu estado de operação:

E(i) = "reativação"
(quando da recuperação do módulo que estava falho).

E(i) = "eleição"
(quando estiver sendo executado o AE)

E(i) = "criação/recuperação"
(quando o BDR estiver sendo criado/recuperado)

E(i) = "normal"
(quando pronto para executar as primitivas oferecidas aos usuários)

Ender-SNL = endereço do SNL local (ativo ou falho)

ISNL-eleita = identificação da ISNL que deve recuperar o SNL/BDR.

Em qualquer momento as ISNL's, ou têm o endereço do SNL ativo, ou supostamente falho.

O SNL é considerado falho por uma ISNL quando:

1) a própria ISNL detetou a falha do SNL após ter enviado uma mensagem e não ter enviado uma mensagem e não ter conseguido resposta em um tempo determinado (TIMEOUT) ;
2) quando tiver recebido uma mensagem de um SNL ou ISNL de outra rede(1) avisando que o SNL local está inativo;
3) quando tiver recebido uma mensagem de uma das ISNL's da rede local tentando saber se as ISNL's de mais alta ordem estão ativas.
4) quando tiver recebido uma mensagem de uma SNL de maior prioridade informando que ela se considera eleita para recuperar o SNL/BDR rede local.

ASSERÇÃO 1

Num dado instante os vetores de estados das ISNL's podem estar configurados em uma das formas abaixo:

1. a) E(i) = "eleição" e Ender-SNL = "indefinido"
   b) outras ISNL's:
      E(j) = "normal" e
      Ender-SNL = "endereço do SNL falho"
      ou E(j) = "eleição" e
      Ender-SNL = "indefinido" ;

2. b) E(i) = "criação/recuperação" e
   Ender-SNL = "indefinido"
   outras ISNL's:
      E(j) = "eleição" e Ender-SNL = "indefinido",
      ou E(j) = "criação/recuperação" e
      Ender-SNL = "indefinido";

3. c) E(i) = "normal e Ender-SNL = "endereço SNL ativo"

outras SNL's:
E(j) = "criação/recuperação" e
Ender-SNL = "indefinido",
ou E(j) = "normal" e Ender-SNL = "endereço do SNL ativo";

Em qualquer dos casos em que o SNL for detetado como falho, a ISNL(i) muda seu estado de "normal" para "eleição" (E(i) = "eleição") e dispara a execução do AE.

Ao final da execução do AE somente uma ISNL será eleita para ativar/recuperar o SNL/BDR na rede local. Se por ventura houver falha da estação(2) do ISNL eleita, o AE deve ser executado novamente por todas as ISNL's locais ativas. Do mesmo modo faz-se necessário quando o BDR não consegue ser criado/recuperado por falta de espaço em disco na estação da ISNL escolhida(3).

ASSERÇÃO 2

Após o AE ter sido concluído com sucesso todas as ISNL's locais ativas estarão com seus estados em "normal" e com endereço da SNL igual ao endereço da ISNL responsável pela ativação/recuperação do SNL, ou seja:

a) E(i) = "normal" e Ender-SNL = "Ender-ISNL-eleita";
b) todas as outras ISNL's:
E(j) = "normal" e Ender-SNL = "Ender-SNL-eleita".

Vejamos agora a descrição estrita do AE que tem como base as referências [6,7].

O critério básico para que uma ISNL seja eleita será vinculado à sua prioridade em relação às outras ISNL's da rede na qual o SNL deve ser ativado/recuperado. Prioridade essa que é estabelecida em função da identificação das estações dentro da rede. Cada módulo no ambiente da rede local tem um número único que o distingue dos demais, variando de 1 a "n", onde "n" é o número máximo de nodos na rede. O de mais alta prioridade é exatamente o módulo "n". A ISNL eleita deve ser aquela do módulo de maior numeração dentre os que estiverem ativos no momento da execução do AE.

Quando uma ISNL(i) descobre falha do SNL (seja qual for a forma - vide considerações anteriores) ela aciona o protocolo de eleição objetivando eleger a si própria, ou outra ISNL, como componente do SN a ativar/recuperar o SNL/BDR na respectiva estação.

O protocolo de eleição executado por qualquer ISNL(i) é dividido em duas partes. Primeiro, a ISNL(i) tenta contatar com todas as outras ISNL's de maior prioridade (módulos de numeração maior que a sua). Se pelo menos uma das ISNL's recebe a mensagem en-

(1) sempre que um componente do SN (SNL ou ISNL), em qualquer rede deteta a falha do SNL de outra rede, este envia uma mensagem para o grupo das ISNL's da rede do SNL falho avisando a queda do mesmo.

(2) fato detetado por esgotamento de temporização.

(3) situação detetada pela ISNL que solicita a ativação/recuperação do SNL/BDR.

viada, atestando que está ativa, a ISNL(i) coloca-se na situação de espera até que uma ISNL de prioridade mais alta torne-se a eleita. Entretanto, passada em espera um certo limite de tempo pré-estabelecido (TIMEOUT), sem que receba alguma mensagem sobre a ISNL eleita, a ISNL(i) reaciona o protocolo de eleição desde o início. De outro modo, se nenhuma das ISNL's de maior prioridade responderem num limite de "T" segundos, a ISNL(i) assegura-se que todas elas estão falhas e considera-se como componente eleito.

Após o cumprimento da primeira etapa, a interface escolhida envia mensagem a cada uma das ISNL's de prioridade mais baixa que a sua, a fim de informar que ela foi a eleita. Ativa o SNL localmente e solicita-lhe, se for a primeira vez (através da primitiva CRIA-BDR) que crie o BDR da rede, ou se por motivo de falha (através da primitiva REC BDR) que recupere o BDR da rede. Dependendo do resultado da execução das primitivas solicitadas, a ISNL(i) envia novamente mensagem às ISNL's de menor prioridade informando se o SNL/BDR foi, ou não, criado/recuperado com sucesso.

As ISNL's de menor prioridade que a ISNL (eleita) ao receberem a mensagem de eleição da interface eleita, mudam seus "estados" de "eleição" para "criação/recuperação" (do SNL/BDR) e aguardam até o recebimento da mensagem informando que o SNL/BDR foi criado/recuperado normalmente, quando só então trocam seus "estados" para "normal" (E(j)= = "normal"). Se após um tempo determinado (TIMEOUT) não receberem tal mensagem, reiniciam o protocolo de eleição. Da mesma forma se receberem uma mensagem da ISNL eleita informando que não existe espaço em disco suficiente para recuperar o BDR, reativam o AE.

4. CONSIDERAÇÕES FINAIS

Maiores detalhes sobre o Servidor de Nomes aqui apresentado, podem ser encontrados na referência [10]. Além dos tópicos tratados neste artigo, estão especificados, em [10], a sintaxe e semântica do protocolo do SN, bem como os aspectos relativos à implementação.

O SN objeto deste trabalho foi desenvolvido na linguagem Pascal MT Plus, no ambiente da REDPUC [1 a 4], entretanto tem portabilidade para ser transportado para outras redes locais. Constitue-se como pioneiro aqui no Brasil e é flexível no sentido de suportar novos serviços para melhor atender às necessidades dos seus usuários.

BIBLIOGRAFIA:

1. GOMES SOARES, L.F.; MENASCÉ,D.A. e outros, "Descrição de Hardware e Software da Rede Local, REDPUC", Anais do XV Congresso Nacional de Informática, Rio de Janeiro, outubro 1982.

2. GOMES SOARES, L.F.; MENASCÉ,D.A., "Um protocolo para redes locais do tipo difusão", Anais do IX Seminário Integrado de Software e Hardware, julho 1982.

3. GOMES SOARES, IERUSALIMSCHY, R., "Especificação de Protocolos através de redes de Petri com temporização - O Protocolo de Acesso a Barra da Rede Local - REDPUC", RBC, Rio de Janeiro, Vol. 3, Nº 3, 1983/1984, pp. 171-193.

4. ZÚNICA DE SPIRITO,P.A., "Um ambiente para implementação de serviços em uma rede local de computadores", Dissertação de mestrado, Departamento de Informática, PUC/RJ, 1984.

5. MENASCÉ, D.A., "Especificação de um protocolo para interligação de Redes locais", Anais do IV Congresso da Sociedade Brasileira de Computação, julho 1984.

6. MENASCÉ, D.A., POPEK, G.J., and MUNTZ, R.R., "A Locking Protocol for Resource Coordination in Distributed Databases", ACM Transactions on Database Systems, Vol.5 Nº 2, pp. 1Ø3-138, junho 198Ø.

7. GARCIA-MOLINA, H., "Elections in a Distributed Computing System", Princeton University - Departament of EECS, dezembro 198Ø.

8. MENASCÉ, D.A. e SCHWABE D., "Redes de Computadores - Aspectos Técnicos e Operacionais", Editora Campus, 1984.

9. OPPEN, Derek C. and DALAL, Yogen K., "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distribuited Envionment", Xerox Corporation, outubro 1981.

10. MEDEIROS, J. Laedio, "Especificação e Implementação de um Servidor de Nomes para uma Rede Local", Dissertação de Mestrado, Departamento de Informática, PUC/RJ, fevereiro 1985.

ESPECIFICACION FORMAL DE PROTOCOLOS DE COMUNICACION.

P. O'CALLAGHAN*, C. RODRIGUEZ**

RESUMEN

En este trabajo, se presenta el problema de la especificación formal de protocolos de comunicación. Se muestra que los métodos clásicos de especificación son inadecuados para este fin, debido en gran parte a la presencia del tiempo como elemento clave en la descripción de un protocolo. Se propone la utilización de un nuevo formalismo que potencialmente puede superar esta dificultad.

ABSTRACT

In this paper we discuss the problem of the formal specification of communications protocols. Classical methods of specification are shown to be inadequate for the purpose, largely due to the presence of time as a key component in protocol descriptions. We propose the use of a new formalism which may potencially be able to overcome this difficulty.

* Profesor de la Universidad Simón Bolívar, Caracas, Venezuela. Ph.D. en Ingeniería de la Computación, Heriot-Watt University, Edimburgo, Escocia. Areas de interés: sistemas de operación, sistemas distribuidos, redes de computadores.

** Profesor de la Universidad Simón Bolívar, Caracas, Venezuela. Ingeniero de la Computación, Universidad Simón Bolívar. Areas de interés: lenguajes de programación, teoría de la computación, sistemas distribuidos.

INTRODUCCION

Los Sistemas de Especificación formal constituyen un area de investigación muy activa dentro de las Ciencias de la Computación. Una gran proporción de estos esfuerzos está dirigida hacia la especificación de programas secuenciales, y en particular hacia las estructuras de datos. Esto es natural, dada la importancia de la programación secuencial en términos de lo que hacen la gran mayoría de los programadores. Sin embargo, hay un creciente interés en las estructuras de control no secuenciales. Esto se debe en parte al hecho de que tales estructuras pueden ser más "naturales" en ciertas circunstancias, y en parte al simple hecho de que el mundo real no es secuencial, y por lo tanto un programa que pretende funcionar bien en su relación al mundo tiene que acomodarse a este hecho. Ejemplos de tales programas incluyen los sistemas de operación, las redes de computadoras, los sistemas tolerantes a fallas, los sistemas de "tiempo real", e inclusive muchos juegos computarizados.

Aquí nos restringimos a una sub-clase de los sistemas no secuenciales (vea [22] para una clasificación más completa), llamados los *sistemas distribuidos*. Para nosotros, un sistema distribuido es aquello en que no hay ningún control centralizado que sirva para sincronizar los diferentes partes autónomas. Así excluimos los sistemas de operación de máquinas aisladas, y las arquitecturas vectoriales, por ejemplo. En particular, nuestra preocupación es con la especificación de los protocolos de comunicación dentro de una red. El problema surgió cuando tuvimos que explicar algunos protocolos a estudiantes de pregrado. Como la especificación de estos sólo existía en lenguaje natural, dió lugar a muchas ambigüedades y hasta "*huecos*" en la definición. Estamos convencidos que un mecanismo formal para estos casos es no sólo deseable, sino necesario.

En este artículo, nos limitamos al caso de la comunicación entre dos nodos "*vecinos*" (es decir, sin nodos intermedios). Aún así, el problema de la especificación es no trivial, y hay que decir de entrada que no lo hemos resuelto. Sin embargo, nos parece interesante presentar lo que hemos hecho hasta ahora.

PROTOCOLOS DE COMUNICACION.

Un *protocolo de comunicación* es una disciplina o convención seguida por dos o más participantes ("*nodos*") para el intercambio eficiente de información a través de un medio de comunicación sujeto a errores (ver por ejemplo [20]). El hecho que hayan errores de transmisión de mensajes, incluyendo pérdidas totales de mensajes, hace que un protocolo tiene que ser un sistema tolerante a fallas. Para estar seguro de que sus mensajes están llegando bien, el transmisor espera "*acusos de recibo*", o "*ACKs*", enviados por el receptor. Esto quiere decir que las fallas de transmisión se detectan por la no llegada del ACK correspondiente. Debido a esto, el transmisor tiene el problema de cuando puede asumir que el ACK no le va a llegar, y retransmitir el mensaje. El momento en que esto se decide es señalado por un evento llamado *timeout*. (Nótese que puede haber varias causas de la ausencia del ACK en el momento de producirse el timeout, que son: el ACK no fué enviado, fué enviado y se perdió, o simplemente se retrasó. El transmisor no tie

ne forma de distinguir entre estas posiblidades).

Está claro, entonces, que toda buena especificación de un protocolo debe incluir el tiempo como elemento básico. Esto es lo que los hace difíciles a especificar, y a la vez interesantes a estudiar.

SOBRE LAS ESPECIFICACIONES.

Entendemos por una *especificación* una descripción o definición de una clase de objetos que lo distingue en forma precisa del resto del universo. En otras palabras, la especificación indica lo que todos los posibles miembros de la clase deben tener en común. Concretamente, en el caso de los protocolos, podemos decir que todas las posibles implementaciones de un protocolo dan todos los miembros de la clase definida por la especificación del mismo. Algunas ventajas de trabajar con buenas especificaciones son:

(1) Poder saber si una implementación cumple o no con su especificación.
(2) Poder demostrar que dos implementaciones son equivalentes.
(3) Poder demostrar propiedades de la especificación en sí, y por lo tanto de toda implementación posible.

Este último punto es tal vez el más interesante, en el sentido de que, si la especificación se hace con una herramienta formal, es decir bien fundada en la matemática o la lógica, nos abre la posibilidad en principio de poder *verificar* la especificación. En otras palabras, podríamos eventualmente determinar si la especificación es "*correcta*" o no, bajo ciertos criterios. Por ejemplo, en una especificación de protocolo, nos interesa saber si permite estados de "*deadlock*", si la secuencia de mensajes aceptados por un receptor es una secuencia inicial de los que el transmisor quiere enviar y que todo mensaje enviado será aceptado después de un número acotado de repeticiones (determinado por la frecuencia de errores en el medio). Además, nos interesan propiedades más universales - por ejemplo que la especificación es completa y que no es ambigua. En este artículo nos concentramos en las técnicas de especificaciones en sí, sin preocuparnos demasiado por los métodos de verificación.

A continuación, presentaremos en forma muy suscinta las características resaltantes de algunas metodologías que se han usado para la especificación de protocolos.

MAQUINAS DE ESTADOS FINITOS.

Los modelos basados en Máquinas de Estados Finitos fueron usados para especificaciones de protocolos desde el mismo momento en que se intentó resolver este problema [1, 2, 3]. Esto se debió, principalmente, a que los nodos se comportan como este tipo de máquinas: en todo instante de tiempo están en un estado diferenciable de todos los demás y ante la ocurrencia de algún evento, se produce una transición determinística a otro estado.

Cuando se usa este tipo de especificaciones, se considera como un estado cada combinación posible de valores de las variables del sistema, lo que excluye la especificación de procesos o entidades con variables con rangos de valores de cardinalidad infinita.

Por otra parte, aún en el caso de conjuntos finitos de valores, el número es estados a considerar crece exponencialmente si se aumenta el rango de valores de algunas de las variables o se añade un nuevo elemento al sistema, lo que hace la especificación intratable por su tamaño. Su aplicabilidad queda entonces restringida a protocolos muy sencillos que controlen un número pequeño de entidades.

Modelos posteriores intentaron paliar estos problemas mediante la compactación de la información de varios estados en un sólo estado. Esto por supuesto, disminuía la cantidad de estados a considerar, pero esta reducción no es lo suficientemente fuerte como para dejar una especificación clara y manejable del problema.

MODELOS BASADOS EN REDES DE PETRI.

Las redes de Petri [4] son el formalismo que más ha sido usado en la especificación de protocolos de comunicación. Esto se debe a que desde su creación han sido muy usadas en la descripción de sistemas de procesos concurrentes que se comunican. En el caso de protocolos, se puede detallar una larga lista de trabajos sobre el uso de esta metodología (ver [5, 6, 7, 8, 9, 10] a manera de ejemplo), pura o con enriquecimiento (Redes de Petri con condicionales [10, 12], con emisión y recepción de mensajes [11], etc). Son más aplicables que las máquinas de estados finitos, ya que en una red finita, donde el número de fichas puede crecer arbitrariamente, se pueden representar infinitos estados. Pero, a pesar de proporcionar facilidades para efectuar verificaciones sobre algunas especificaciones (*volveremos sobre este punto luego*), al intentar especificar problemas más complicados que los del Bit Alternado, el tamaño de la red resultante hace que la especificación sea inmanejable.

ESPECIFICACIONES USANDO LOGICA.

Otro formalismo usado en la especificación de protocolos, es la lógica proporcional. Basados en ella, Chen y Yeh [13] desarrollaron un lenguaje llamado EBS (*Event Based Specification Language*), que sirve para formular aseveraciones que describen las secuencias válidas de eventos que puede generar un conjunto de procesos. Una característica resaltante es la no utilización de relojes globales que regulen el comportamiento de todo el sistema. Definen, sin embargo, una relación de orden parcial, denotada por ' $\rightarrow$ ', para representar la precedencia entre eventos. Los elementos incomparables bajo esta relación son llamados concurrentes. Utilizan además, para formular sus aseveraciones, la relación de causalidad: a $\Rightarrow$ b significa que la ocurrencia del evento 'a', causa la ocurrencia en el futuro del evento 'b'. Nótese que si a $\Rightarrow$ b, entonces en toda secuencia válida de eventos a $\rightarrow$ b, pero el contrario no es necesariamente cierto.

LOGICA TEMPORAL.

Otro enfoque es la utilización de Lógica Temporal [14], constituída por elementos de la Lógica Clásica de Primer Orden mas los operadores temporales $\Box$, , y *Until*. Para interpretar estos operadores, se supone que el sistema está descrito por un conjun

to de secuencias de estado, que son sus posibles comportamientos. La fórmula $\Box A$ ("*henceforth* A") es cierta si A es cierta en el estado actual y en todos los estados futuros. Si A es cierta, entonces $\Box A$ es cierta en el estado actual o en algún estado futuro. A *until* B es cierta si A es cierta en todos los estados hasta el primer estado donde B es cierta.

Tanto EBS como la Lógica Temporal son resultados interesantes dentro del campo de la especificación de protocolos, pero tienen sus limitaciones. Facilitan la descripción de las secuencias correctas de eventos o estados de un sistema, pero no permiten decir en forma clara y precisa como lograr la producción de esas secuencias.

LENGUAJES DE PROGRAMACION DE ALTO NIVEL

Otra idea dentro del campo de la especificación de protocolos es la utilización de lenguajes de alto nivel para representar las entidades del sistema. Tiene la ventaja de describir en forma precisa la secuencialidad de las acciones que debe efectuar cada elemento del sistema y las estructuras usadas para manejar la información, pero no siempre proveen mecanismos claros para describir la concurrencia.

Uno de los lenguajes empleados para especificar protocolos es AFFIRM [15] , que es usado principalmente para hacer especificaciones algebraicas. Para especificar un protocolo en AFFIRM, se supone que cada estación es un tipo de datos abstracto. La ocurrencia de un evento se representa como la ejecución de una de las operaciones posibles sobre el tipo de datos. Una de sus principales desventajas es que no provee mecanismos para relacionar o sincronizar entidades separadas. Esto imposibilita hacer la especificación de una red, pudiendose sólo describir sus componentes.

Hemos experimentado también con el lenguaje CSP de Hoare [16] CSP tiene buenas estructuras de control para representar la concurrencia y el no determinismo. En particular, la comunicación entre procesos en CSP puede ser usado para modelar las actividades de nodos en una red. Sin embargo, no es propiamente un lenguaje de especificación, sino de programación, por lo tanto al usarlo, obtenemos sólo un miembro de la clase a ser especificada.

TECNICAS DE VERIFICACION

Una característica altamente deseable de un formalismo de especificación es la posibilidad de probar propiedades como las indicadas en la Introducción. En el espacio disponible aquí, no podemos profundizar sobre este aspecto. Sólo haremos las siguientes observaciones sobre algunos de los formalismos de especificación referenciales anteriormente.

Las técnicas de verificación más comunes, cuando se especifica usando Redes de Petri o metodologías afines, consisten en la generación exhaustiva de todos los estados alcanzables por la máquina que describe la especificación. Para ello, existen herramientas automatizadas, tales como OGIVE/OVIDE [18] y CESAR [17], que pueden realizar análisis de alcanzabilidad de estados dentro de una red acotada. Esto resulta ser suficiente para

probar muchas propiedades de la especificación, incluyendo "*liveness*" [23] y ausencia de "*deadlocks*".

En el caso de los lenguajes de programación de alto nivel, las verificaciones pueden hacerse mediante las técnicas clásicas de pruebas de invariantes. Este método tiene la desventaja de necesitar de demostradores automáticos de teoremas para poder realizar las verificaciones, y las herramienta mecanizadas disponibles aún tienen un dominio de aplicación muy limitado.

En la próxima sección, presentamos algunas ideas sobre el uso de otro formalismo de especificación que ofrece ciertas ventajas en este aspecto.

SPC

Una metodología de especificación que nos parece muy interesante es el Lenguaje Formal para Especificación de Procesos Concurrentes [21]. En este formalismo cada proceso se representa como un conjunto finito de reglas de reescritura con eventos. Este conjunto de reglas se puede interpretar como la descripción finita de un sistema (*posiblemente infinito*) de transición de estados. Bajo esta óptica, cada una de las reglas representa un posible cambio de estado del proceso, junto con la realización de un conjunto de comunicaciones (*posiblemente vacio*). Por ejemplo, si queremos describir el proceso Pila, que recibe números naturales por una puerta de entrada y los emite por una puerta de salida según el orden LIFO (*último en entrar, primero en salir*), el conjunto de reglas sería:

Estado Inicial	p( )	
p(W)	$\xrightarrow{\{\alpha.e(N)\}}$	p(cons(W,N))
p(cons(W,N))	$\xrightarrow{\{s.\alpha(N)\}}$	p(W)

Las comunicaciones en este formalismo se denotan por términos de la forma p1.p2(T), donde p1 es la puerta emisora, p2 la receptora y T el elemento que se transmite. En caso que el emisor o el receptor sea el ambiente donde se está ejecutando el programa, se coloca como puerta el símbolo 'α'.

La primera de las reglas especifica que si el contenido de la pila es la secuencia W, y llega un entero N como entrada, el resultado será una pila con N como elemento tope y la secuencia W como resto. La segunda dice que si la pila tiene como elemento tope N y resto W, puede emitir por la puerta de salida a N y quedar con W como contenido.

Una vez que se tienen las descripciones de un conjunto de procesos, se puede construir la descripción de un sistema de procesos comunicantes mediante la combinación de las especificaciones de los procesos constituyentes y el uso de los operadores ||, +, y - . El operador || (*paralelismo*), recibe como entrada las especificaciones de dos procesos P1 y P2 y produce como salida las reglas del proceso P que resulta al ejecutar P1 en paralelo con P2. El operador '+' toma como entrada las reglas que describen un proceso P

y un par de puertas (*una de entrada y otra de salida*) y produce la descripción del proceso que resulta a partir de P conectando las puertas indicadas. El operador'-' (*restricción*) produce la descripción de un proceso que resulta de "*ocultar*" las comunicaciones a través de un canal C, dentro de un proceso P, que le son dados como parámetros.

A contiuación usaremos esta metodología para especificar el protocolo del Bit Alternado. Supondremos la red constituida por cuatro procesos: el Transmisor, el Receptor, un proceso para representar el canal de comunicación simple del transmisor al receptor, y un proceso para representar el canal en sentido inverso. Representamos los canales como procesos porque, desde el punto de vista del protocolo, son entes activos que pueden distorsionar o perder parte de los mensajes que circulen por ellos. Para los efectos de la especificación, la distorsión y la pérdida de mensajes se consideran eventos equivalentes.

Transmisor

Este proceso tiene tres puertas, dos de entrada y una de salida. A través de la puerta de entrada 'α.e' recibe, desde el anfitrión, los mensajes a transmitir; por la puerta de salida 't.α' los transmite al exterior y por la puerta 'α.a' recibe ACKs. El conjunto de reglas que describen al proceso es:

Estado Inicial	i(0)	
i(X)	$\xrightarrow{\{\alpha \cdot e(M)\}}$	r(X,M))
r(X,M)	$\xrightarrow{\{t.\alpha(c(X,M))\}}$	1(X,M)
1(X,M)	$\xrightarrow{\{\alpha.a(X)\}}$	i(inv(X))
1(inv(X),M)	$\xrightarrow{\{\alpha.a(x)\}}$	r(inv(X),M)

Receptor

Tendrá dos puertas de salida y una de entrada. A través de 'α.s1' recibe mensajes que pasa al anfitrión usando la puerta 'o.α' y envía ACKs por la puerta 'i.α'.

Estado Inicial	e(0)	
e(X)	$\xrightarrow{\{\alpha.s1(c(X,M))\}}$	k(X,M)
k(X,M)	$\xrightarrow{\{o.\alpha(M).i.\alpha(X)\}}$	e(inv(X))
e(inv(X))	$\xrightarrow{\{\alpha.s1(c(X,M))\}}$	m(inv(X))
m(inv(X))	$\xrightarrow{\{i.\alpha(X)\}}$	e(inv(X))

Canal del Transmisor al Receptor (Proceso C1):

Recibirá mensajes a través de una puerta de entrada, que en forma no determinística emitirá por la puerta de salida 's.α', o no.

Estado Inicial W

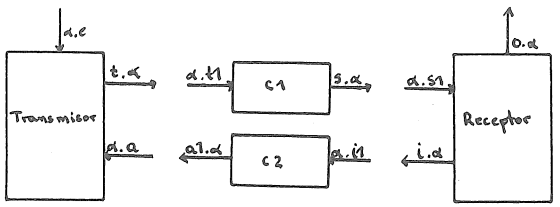
W	$\xrightarrow{\{\alpha.t1(X)\}}$	f(X)
f(X)	$\xrightarrow{\{s.\alpha(X)\}}$	W
f(X)	$\xrightarrow{\{\ \}}$	W

Canal Del Receptor al Transmisor (Proceso C2)

Estado Inicial d

d	$\xrightarrow{\{\alpha.i1(X)\}}$	h(X)
h(X)	$\xrightarrow{\{a1.\alpha(X)\}}$	d
h(X)	$\xrightarrow{\{\ \}}$	d

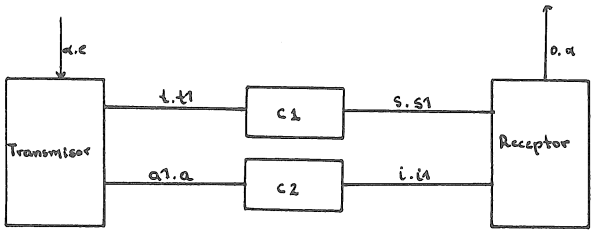
Hasta el momento tenemos las especificaciones aisladas de cuatro procesos. Gráficamente se puede ver de la siguiente forma:



Para tener la especificación del sistema de procesos que describa el comportamiento de la red, se debe calcular el proceso que resulta de colocar en paralelo los cuatro procesos, conectar los pares de puertas ('t.α', 'α.t1'),(s'.α','s1.α'),('α.a','a1.α'), ('i.α', 'α.i1') y *"ocultar"* las comunicaciones por las puertas 't.α','α.t1','s.α','s1.α','α.a', 'a1.α','i.α','i1.α'. Una expresión que representa lo anterior es:

$$((t \,\|\, C1) + 't.t1 - t.\alpha - \alpha.t1) \,\|\, ((R \,\|\, C2) + i1.i - \alpha.i - i1.\alpha)) + s.s1 + a.a1 - s.\alpha - \alpha.s1 - a1.\alpha - \alpha.a$$

El sistema resultante es:



Para poder estudiar con comodidad la aplicabilidad de este formalismo en la especificación de protocolos, se ha desarrollado un programa en PROLOG [19] que realiza evaluaciones de expresiones de este álgebra de procesos. Paralelamente, se están estudiando mecanismos para realizar verificaciones de propiedades de la especificaciones de redes de procesos que resultan de evaluar estas expresiones, con la finalidad de aplicarlas al caso específico de los protocolos. Estos estudios se encuentran avanzados y proximamente publicaremos los resultados obtenidos.

CONCLUSIONES

El problema de la especificación formal de los protocolos de comunicación es difícil. Las técnicas tradicionales de especificación de programas secuenciales, e incluso de sistemas de procesos concurrentes, no satisfacen los requerimientos necesarios para especificar un protocolo en forma cabal. Específicamente, carecen de mecanismos para describir restricciones de tiempo, lo cual es imprescindible en este contexto, y en muchos casos sus facilidades de verificación están limitadas tanto por el tamaño de lo especificado como por las propiedades que se pueden estudiar.

En este trabajo, hemos propuesto utilizar el formalismo SPC [21] , no empleado previamente para este problema. Bajo este formalismo, podremos tener una descripción finita de la máquina de estados que define el protocolo, con la ventaja adicional de poder hacer verificación de muchas propiedades de la especificación, incluyendo "*liveness*", ausencia de "*deadlocks*" y otros invariantes sobre secuencias de eventos.
Estamos trabajando ahora en el problema de como introducir el tiempo en SPC, para así contar con una herramienta más aplicable a este problema. Esperamos que ella pueda servir también para problemas relacionados, tales como sistemas de tiempo real, sistemas tolerantes de fallas, etc.

(1) Kawashima, Futani y Kand: "*Functional Specification of Call Processing by State Transition Diagram*", IEEE Trans. Comm. Tech., Vol COM19, Oct 1971.

(2) Birke: "*State Transition Programming Techniques and Their Use in Producing Teleprocessing Device Control Programs*", IEEE Trans. Comm., Vol. COM20, Nro 3, Junio 72.

(3) Bochmann C.: "*Communications Protocols and Error Recovery Procedures*", ACM Operating System Review, Vol 9, Nro. 3, Julio 1975.

(4) Petri C.A.: "*Kommunikation mit Automaten*", Schriften des Rheinish-Westfalischen Institutes fur Instrumentelle Mathematik an der Universitat Bonn, Heft 2, Bonn, W. Germany 1962; Tech. Fep. BADC-TR-65-337.

(5) Keller, R. M. "*Formal Verification of Parallel Programs*",CACM, Vol. /, 1976.

(6) Merlin, P. "*Specification and Validation of Protocols*", IEEE Trans. Comm. Vol. COM27, Nro. 11, Nov 1979.

(7) Ellis: "*Consistency and Correctness of Duplicate Database Systems*", Proc. of Sixth ACM Symp. in Operating Systems Principles, Nov 1977.

(8) Barlett, Scantlebury y Wilkinson: "*A note on Reliable Full-Duplex Transmissions over Half-Duplex Links*", CACM, Vol. 12, p. 260, 1969.

(9) Yoeli y Barzilai: "*Behavioral Descriptions of Communication Switching Systems Using Extended Petri Nets*", Digital Processes, Vol 3, 1977.

(10) Berthelot, Terrat: "*Petri Net Theory for the Correctness of Protocols*", IEEE Transactions on Communications. Vol Cc 30, Nro 12, Dic 1982.

(11) Jurgensen, Vuong: "*Formal Specification and Validation of ISO Transport Protocols Components Using Petri Nets*", ACM SIGCOMM, Sept. 1984.

(12) Courtiat, Ayache, Algayres: "*Petri Nets are Good for Protocols*", ACM SIGCOMM, Sept. 1984.

(13) Chen, Yeh: "*Formal Specification and Validation of Distributed Systems*" IEEE Proc. 2nd. Intl. Conf. on Distributed Computing Systems.

(14) Schwartz, Mellar-Smith: "*From State Machines to Temporal Logic: Specification Methods for Protocol Verification*", IEEE Trans. Comm. Vol COM 30, Nro 12, Dic 1982.

(15) Sunshine, Thompson, Brickson, Gerhart, Schabe: "*Specification and Validation of Communication Protocols in* AFFIRM *Using State Transiton Models*", IEEE Trans. on Soft. Eng., Vol SE8, Nro 5, Sept. 1982.

(16) C.A.R. Hoare: *"Communicating Sequential Processes"*, CACM, Agosto 1978.

(17) Queille: *"The CESAR System: A Computer Aided Design and Certification System for Distributed Applications"*. Proc. 2nd Int. Conf. on Distributed Computing Systems, Abril 1981.

(18) Montel: *"OVIDE: A software Package for the Validation of Systems Represented by Petri Net Pased Models"*, 4th. European Workshop on Application and Theory of Petri Nets, Toulouse, Sept. 1980.

(19) Roussel Ph.: *"PROLOG: Manuel de Référence et d'Utilisation"*, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille.

(20) Tanenbaum A.: *Computer Networks"*, Prentice Hall Software Series, 1981.

(21) Pereira J. M.: *"Procesos Communicants"; Un Language Formal et sus Models, Problemes d'Analyse"*, Thêse de 3'eme Cycle, Junio 1984, Université de Grenoble.

(22) O'Callaghan y Foulk, *"Formal Description of Parallel Structure in AIDS"*, IEE Trans. on Computers and Digital Techniques, Marzo 1980, pp. 55-63.

(23) Peterson J.: *"Petri Net Theory and Modelling of Systems"*, Prentice-Hall Inc.

UMA IMPLEMENTAÇÃO DE UM PROTOCOLO DE TRANSFERÊNCIA DE ARQUIVOS PARA REDES DE COMPUTADORES

ORIVALDO DE LIRA TAVARES*

DANIEL SCHWABE**

SUMÁRIO

Este artigo apresenta aspectos importantes da implementação de um protocolo de transferência de arquivos de uso geral, com independência dos sistemas de arquivos envolvidos na transferência e com opção de reinício automático da transferência.

Este protocolo situa-se acima do nível de transporte, ou seja, nos níveis de sessão, apresentação e aplicação da arquitetura para Interconexão de Sistemas Abertos proposta pela ISO.

ABSTRACT

This paper presents important aspects of the implementation of a file transfer protocol of general use, allowing for file system independence and with an automatic transfer restart option.

This protocol is situated above the transport layer, that is, in the session, presentation and application layers within the Open Systems Interconnection architecture proposed by ISO.

* Professor Assistente da Universidade Federal do Pará. Mestre em Informática pela PUC-RJ, em programa de doutorado na PUC-RJ. Áreas de interesse: Redes de Computadores, Engenharia de Software, Inteligência Artificial.

**Professor Assistente da Pontifícia Universidade Católica-RJ.Doutor em Ciência da Computação pela UCLA. Áreas de interesse: Redes de Computadores, Protocolos, Sistemas Distribuídos, Inteligência Artificial.

1. INTRODUÇÃO.

Um protocolo de transferência de arquivo (PTA) é um conjunto de regras que, uma vez obedecidas, permitem a transferência de arquivos entre os diversos nodos (computadores usuários) de uma rede de computadores.

Um PTA faz basicamente uma operação de cópia de arquivos entre sistemas de computação da(s) rede(s). Assim sendo, é conveniente que ofereça a possibilidade de compatibilizar as diferenças existentes entre as representações dos arquivos destes sistemas de computação. Outra facilidade interessante é a capacidade de reiniciar uma operação de transferência, após a ocorrência de falha, sem perder uma parcela significante do trabalho já executado.

A implementação da transferência de arquivos entre nodos de características distintas é feita através da criação de um arquivo cujas características são adaptáveis aos nodos participantes da transferência. Este arquivo é denominado arquivo virtual e é definido por um conjunto de atributos que o caracterizam. Para que a transferência de um arquivo se realize, o PTA converte o arquivo a ser transferido para o formato do arquivo virtual, copia o arquivo virtual para o nodo destino da transferência, e converte o arquivo virtual para um formato de arquivo compatível com as características do nodo destino.

Apresentar-se-á aqui um PTA que implementa as facilidades referidas anteriormente e, devido ao seu projeto modular, permite sua implementação em versões que comportem o sub-conjunto de facilidades ideal para as exigências dos usuários.

Para haver a transferência de arquivos entre dois nodos de uma rede é necessário que uma versão do PTA controle cada extremidade da transferência. O PTA que fica no lado acessado pelo usuário será denominado PTA-PRIMÁRIO. O PTA do outro extremo será o PTA-SECUNDÁRIO. Quando a operação de transferência estiver em andamento, usar-se-á a denominação PTA-EMISSOR e PTA-RECEPTOR para os protocolos que estão no lado emissor e no lado receptor do arquivo, respectivamente.

A ISO (Organização Internacional para a Padronização) definiu em

[5] um Modelo de Referência para a Interconexão de Sistemas Abertos. Este Modelo de Referência indica uma arquitetura de 7 (sete) níveis (físico, enlace de dados, rede, transporte, sessão, apresentação e aplicação) para a estruturação das funções de interconexão de sistemas abertos.

O PTA apresentado neste artigo foi definido sobre o nível de transporte e implementa as funções necessárias dos níveis de sessão, apresentação e aplicação.

2. DESCRIÇÃO DO PTA.

Para que várias transferências possam ser realizadas simultaneamente, este protocolo possui estruturas de dados que guardam as informações necessárias para delimitar o contexto de cada transferência.

Antes de qualquer outra coisa, o usuário do PTA deve abrir ou delimitar um contexto de transferência, ou seja, deve fornecer ao PTA as informações necessárias para o controle e efetivação da transferência.

Após aberto o contexto de transferência, o usuário pode solicitar as cópias (transferências) dos arquivos que desejar, fornecendo os atributos do arquivo virtual que será usado em cada cópia. Uma vez executadas todas as cópias desejadas, o usuário encerra o contexto de transferência, liberando os recursos que lhe foram alocados. Além disto, o PTA permite que o usuário consulte os atributos vigentes no contexto de transferência, aborte uma cópia em execução ou aborte o contexto da transferência.

A estrutura do PTA possui dois níveis responsáveis pela execução das funções. O nível inferior é o nível que executa as funções do nível de sessão, do Modelo de Referência da ISO, necessárias ao PTA, por isto, este nível é denominado PTA-SESSÃO. O PTA-SESSÃO realiza os seus serviços controlado pelo nível superior. Este nível superior, chamado de PTA-APLICAÇÃO, engloba as funções dos níveis de apresentação e aplicação do Modelo de Referência da ISO e executa os seus serviços quando solicitados pelos usuários do protocolo.

O nível de sessão do PTA deve permitir que o nível superior abra,

encerre ou aborte uma sessão de comunicação, envie ou receba mensagens; reinicie a operação de transferência; e envie marcas de reinício para poder reiniciar a operação de transferência com mínima perda de trabalho.

O PTA-APLICAÇÃO deve permitir que o usuário abra, encerre ou aborte um contexto de transferência; defina o conjunto de atributos que caracterizam o arquivo virtual a ser usado na transferência; copie o arquivo do PTA-EMISSOR para o PTA-RECEPTOR definindo, a priori, os atributos desta operação (nomes dos arquivos do emissor e no receptor, quem vai ser o PTA-EMISSOR, etc.); e aborte a operação de cópia se for necessário.

Descreve-se a seguir alguns detalhes das funções executadas em cada nível do PTA.

2.1. FUNÇÕES DO PTA-SESSÃO

O PTA-SESSÃO realiza as seguintes funções:

- Abertura da sessão de comunicação.
  Esta função é executada pelos níveis de sessão dos PTAs envolvidos na transferência. Para que ela seja realizada o PTA-APLICAÇÃO deve fornecer as informações necessárias para a abertura da sessão.

- Troca de mensagens.

  Os PTAs-SESSÃO executam as trocas de mensagens entre os PTAs-APLICAÇÃO envolvidos na transferência.

- Reinício da transferência.
  Quando solicitado pelo PTA-APLICAÇÃO o PTA-SESSÃO se reposiciona para o reinício da transferência e avisa ao PTA-APLICAÇÃO o ponto a partir de onde a transferência deve reiniciar.

- Controle do reinício.
  A intervalos regulares, o PTA-APLICAÇÃO pede ao PTA-SESSÃO que uma marca de reinício seja inserida no fluxo das mensagens de dados. Os PTAs-SESSÃO, então, guardam dados destes pontos de reinício e, ocorrendo um pedido de reinício, os dados do último ponto de reinício são recuperados e comunicados ao PTA-APLICAÇÃO.

- Encerramento da sessão de comunicação.
  Uma vez solicitado pelo PTA-APLICAÇÃO, o PTA-SESSÃO executa o encerramento da sessão de comunicação liberando todos os recursos alocados para a sessão.

- Aborto da sessão de comunicação.
  O PTA-SESSÃO pode encerrar prematuramente uma sessão de comunicação, quando for solicitado pelo PTA-APLICAÇÃO.

2.2. FUNÇÕES DO PTA-APLICAÇÃO.

O PTA-APLICAÇÃO executa as seguintes funções:

- Abertura do contexto de transferência.
  Esta função é executada pelos PTAs-APLICAÇÃO envolvidos na transferência. Abrir o contexto de transferência é o primeiro passo para a execução da transferência de arquivo.

- Criação do arquivo virtual.
  Quando solicitada pelo usuário, esta função é executada através da negociação entre os PTAs-APLICAÇÃO da transferência. Os atributos que caracterizam o arquivo virtual devem ser aceitos por todos os PTAs envolvidos na transferência.

- Cópia do arquivo.
  Após o usuário solicitar a cópia do arquivo fornecendo os atributos da operação, os PTAs-APLICAÇÃO negociam os atributos da operação e a executam se concordarem com os seus atributos.

- Aborto da operação de cópia.
  O usuário pode pedir para o PTA-APLICAÇÃO abortar a operação de transferência vigente.

- Recuperação dos atributos vigentes no contexto de transferência.
  O PTA-APLICAÇÃO pode recuperar e comunicar ao usuário todos os atributos vigentes no contexto de transferência, tanto os atributos da operação de cópia atual, como os atributos do arquivo virtual ou os atributos do contexto de transferência.

- Encerramento do contexto de transferência.
  Após executar todas as operações desejadas o usuário deve pedir ao PTA-APLICAÇÃO o encerramento do contexto de transferência. Neste instante, o PTA-APLICAÇÃO libera todos os recursos alocados para o contexto de transferência.

- Aborto do contexto de transferência.
  Esta função é executada pelo PTA-APLICAÇÃO quando o usuário solicita o encerramento prematuro do contexto de transferência.

3. DESCRIÇÃO DA IMPLEMENTAÇÃO.

O PTA descrito no item 2 foi implementado através de processos independentes que se comunicam por troca de mensagens. Basicamente são necessários dois processos distintos denominados PTA-SESSÃO e PTA-APLICAÇÃO.

O PTA-SESSÃO é um processo que executa todas as funções pertinentes ao nível de sessão conforme a descrição feita no item 2.1.

O PTA-APLICAÇÃO executa as funções relacionadas no item 2.2. Quando este processo faz parte do PTA-PRIMÁRIO, além das suas funções normais, ele interage com o usuário do PTA.

3.1. ESTRUTURA MODULAR.

A implementação do PTA possui um módulo de INICIALIZAÇÃO, responsável pela inicialização das estruturas de dados do PTA, um módulo PTA-APLICAÇÃO que após ativado será o processo PTA-APLICAÇÃO, e um módulo PTA-SESSÃO que constituirá o processo PTA-SESSÃO. A figura 1 mostra os módulos componentes do PTA.

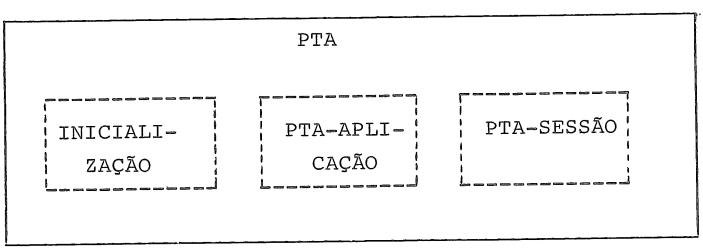


Figura 1 - Módulos componentes do PTA.

3.1.1. ESTRUTURA MODULAR DO PTA-APLICAÇÃO.

O PTA-APLICAÇÃO possui um módulo chamado CONTROLE-COMUNICAÇÃO que recebe e encaminha ao módulo apropriado todas as mensagens destinadas ao PTA-APLICAÇÃO.

O módulo CONTROLE-COMUNICAÇÃO pode enviar cada mensagem recebida para um dos seguintes módulos:

- CRIA-ARQUIVO: Cria um arquivo virtual.
- COPIA: Executa a operação de cópia do arquivo.
- RECUPERA-ATRIBUTOS: Recupera os atributos vigentes dentro do contexto de transferência, comunicando-os ao usuário.
- ABORTA-OPERAÇÃO: Termina prematuramente a operação atual de cópia.
- ENCERRA-TRANSFERÊNCIA: Encerra o contexto de transferência após a realização de todas as operações desejadas.
- ABORTA-TRANSFERÊNCIA: Encerra prematuramente o contexto de transferência.

3.1.2. ESTRUTURA MODULAR DO PTA-SESSÃO.

O PTA-SESSÃO possui um módulo chamado RECEPÇÃO-MSG que recebe e encaminha ao módulo apropriado as mensagens destinadas ao PTA-SESSÃO.

O módulo RECEPÇÃO-MSG pode enviar cada mensagem recebida para um dos seguintes módulos:

- ABRE-SESSÃO: Abre uma sessão de comunicação.
- PROCESSA-MARCA: Envia e recebe as marcas de reinício.
- REINICIA: Reinicia a operação de cópia do arquivo.
- PREPARA-MSG: Prepara as mensagens trocadas entre os PTAs-APLICAÇÃO.
- ENCERRA-SESSÃO: Encerra a sessão de comunicação.
- ABORTA-SESSÃO: Encerra prematuramente a sessão de comunicação.

4. CONCLUSÕES.

A idéia de implementar este protocolo, (referenciado a seguir como PTA), em versões que realizem sub-conjuntos dos serviços disponíveis no PTA completo, permite que seja escolhido o melhor sub-conjunto para cada tipo de usuário. Desta forma otimiza-se o desempenho do protocolo uma vez que a implementação conterá somente o mínimo necessário para o serviço requerido.

O projeto do PTA foi inspirado nas características do FTP (File Transfer Protocol) da ECMA e no FTP proposto pela ISO. Assim, as facilidades oferecidas por eles são semelhantes. A diferença básica está na arquitetura de comunicação. Enquanto que os FTPs da ECMA e da ISO usam a arquitetura padrão da ISO, ou seja, são implementados sobre o nível de apresentação da rede, o PTA foi implementado diretamente sobre o nível de transporte da rede. Apesar disso, a arquitetura do PTA obedeceu ao padrão da ISO, não sendo definido explicitamente apenas o nível de apresentação, cujas funções foram embutidas no nível de aplicação para aumentar a eficiência do protocolo.

Outro fator que influencia no aumento do desempenho deste protocolo é a liberdade que se oferece ao usuário, permitindo-lhe a escolha do melhor conjunto de atributos, (atributos do arquivo virtual e atributos da operação de cópia), para que a operação do PTA seja executada com eficiência máxima.

Quanto ao uso das primitivas do nível de transporte, o PTA-SESSÃO exige que o nível de transporte faça a sequenciação e a temporização das mensagens. Eventuais erros de sequenciação das mensagens de dados são resolvidos pelo módulo COPIA através da solicitação de reinício da operação de transferência, quando a opção de reinício é usada.

A implementação do PTA-APLICAÇÃO e PTA-SESSÃO em processos distintos facilitou a modularização do projeto, uma vez que as funções de cada processo estão bem definidas. Esta modularização permite que muitas experiências possam ser feitas com os níveis do PTA , dado que alterações em um nível não repercutem no outro.

É incerto se a forma de implementação usada neste PTA é a mais eficiente, contudo, este projeto oferece os mecanismos necessá -

rios para atender as principais exigências de transferência de arquivos. O mais importante é que este PTA oferece um ambiente de experiência onde é possível estudar o comportamento e a eficiência da implementação de diversos sub-conjuntos deste protocolo e, por conseguinte, diversos tipos de PTAs.

O PTA descrito neste artigo foi implementado na PUC/RJ, na Rede Local de Computadores - REDPUC. O código gerado pelo Pascal MT+86 para o protocolo, sem os módulos de aborto da transferência, aborto da operação e recuperação de atributos, foi de 64 kbytes. Foi usado um núcleo de comunicação para efetuar o escalonamento e trocar mensagens entre os processos do PTA.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] A Network Independent File Transfer Protocol; HLP/CP(78)1; Dec, 12/77.

[2] Bucciarelli, P.; Poublan A.; et al; ECMA Virtual File Protocol - an Overview.

[3] ECMA; Standard ECMA-85 - Virtual File Protocol; September 1982.

[4] International Standards Organization; ISO/TC97/CS16 WG5, OPEN SYSTEMS INTERCONNECTION; Project 97.16.5; Part I - General Description, N 1190; Part II - The Virtual Filestore, N1222; Part III - The File Service Definition, N1223, Part IV - The File Protocol Specification, N1224 ; May 1983.

[5] International Standards Organization; Reference Model for Open Systems Interconnection; ISO/TC97/SC16/N227; 1978.

[6] Menascé, D.A.; Schwabe, D.; Redes de Computadores - Aspectos Técnicos e Operacionais; Editora Campus, 1984.

[7] Tavares, O.L.; Protocolos de Transferência de Arquivos para Redes de Computadores, Dissertação de Mestrado - Deptº Informática - PUC/RJ; 31/5/84.

Wanderley LOPES DE SOUZA

GRC/DSC/Universidade Federal da Paraíba
(trabalho realizado junto ao IRO/Université de Montrèal com auxílio fornecido pelo CNPq)

Sumário

O uso de técnicas de descrição formal, para a especificação de software de comunicação, tem sido uma constante fonte de preocupações para os projetistas da área em questão. O objetivo desse artigo é realizar uma análise de alguns trabalhos, que sugerem o emprego de Prolog em diversas fases do desenvolvimento de protocolos, e procurar estabelecer um modelo em Prolog, para a especificação de serviços e protocolos, que facilite as tarefas de validação necessárias ao bom desenrolar do projeto.

1. Introdução

Há pelo menos três passos a serem seguidos no desenvolvimento de software de comunicação:

(a) compreender o problema e formular possíveis metodologias para a sua solucão;

(b) especificar a solução de forma não ambigua, utilizando, por exemplo, uma especificação formal;

(c) implementar a especificação através de uma linguagem de programação.

A conformidade entre essas diferentes etapas pode ser assegurada através de métodos de validação, que variam da verificação formal à execução simbólica e da simulação ao teste. Realizada a última etapa, pode-se dizer que a implementação obtida satisfaz à correspondente especificação, de acordo com as atividades de validação empregadas.

O uso de técnicas baseadas em lógica, para descrever a solução de um problema, apresenta algumas vantagens em relação ao uso de linguagens convencionais:

(a) um mapeamento mais direto entre problema e programa;

(b) o programa pode ser dividido em duas partes: um componente **lógico**, especificando o conhecimento necessário à solução do problema, e um componente de **controle**, definindo como usar esse conhecimento para a resolução do problema;

(c) o programa é de fácil modificação e/ou extensão.

Certas características da linguagem de programação lógica Prolog e o caráter lógico presente nos protocolos de comunicação tem induzido pesquisadores a pensar numa nova aplicação para essa linguagem. Nesse trabalho, após uma apresentação sumária de Prolog e de algumas de suas propriedades, o seu emprego nas fases de especificação e validação de serviços e protocolos é discutido.

2. Linguagem de programação Prolog

Prolog é uma linguagem de programação prática, baseada num modelo próximo ao da programação lógica e na interpretação procedural de Kowalski para cláusulas Horn [Kowa 79].

A principal unidade sintática de Prolog é o **termo**. Um termo pode ser uma constante (inteiro, real ou átomo), uma variável ou um termo composto. O primeiro caráter do identificador de um átomo deve ser uma letra minúscula, enquanto que o de uma variável deve ser uma letra maiúscula. Um **termo composto** consiste de um identificador (iniciado com letra minúscula) seguido de uma lista de **argumentos**. Cada argumento já é em si um termo.

De acordo com a sintaxe apresentada em [ClMe 81], as cláusulas Horn são escritas da seguinte forma:

conclusao :- condicao1, condicao2, ..., condicaoN.

onde conclusão e condições são termos compostos do tipo:

termo (arg1, arg2, ..., argM)

A **cláusula** acima define uma **relação** entre conclusão e condições, cuja interpretação é a seguinte: **conclusao**, que é a **cabeça** da cláusula, será verdadeira se (":-") **condicao1** e (",") **condicao2** e ..... e **condicaoN**, que constituem o **corpo** da cláusula, forem verdadeiras. O símbolo "." indica o fim da cláusula.

Uma cláusula sem corpo corresponde a uma **asserção** e será sempre verdadeira. Uma cláusula sem cabeça é interpretada como uma **meta** (ou questão) e nessa sintaxe deve ser iniciada com o símbolo "?-".

Um programa em Prolog é constituido de um conjunto de cláusulas Horn. A execução desse programa corresponde a uma prova, ou sucessão de deduções, de uma dada **meta global** (?- questao1, questao2,....., questaoK.). Tal prova é realizada através de uma pesquisa sequencial no programa, associada a "**backtracking**", visando a unificação das submetas, presentes na meta global, às cláusulas desse programa. O resultado da execução pode ser **sucesso ou falha**. No caso de sucesso, os valores dos parâmetros presentes na meta global constituem a saída da execução.

Em Prolog, o grau de não determinismo, presente no modelo de programação lógica de [Kowa 79] é reduzido. Por questões de simplicidade e eficiência, submetas e cláusulas são escolhidas na pesquisa sequencial, da esquerda para a direita e de cima para baixo respectivamente.

A lista é uma das principais estrutura de dados em Prolog. Uma lista é uma sequência de elementos ordenados, que pode ter qualquer comprimento. Os elementos de uma lista podem ser termos ou listas. Uma lista é frequentemente expressa utilizando-se o operador "|", os delimitadores "[" e "]", da seguinte forma:

[cabeca|cauda]

onde **cabeca** é sempre um elemento e **cauda** uma lista. Uma lista vazia é representada por [].

Estruturas de dados, predicados e estruturas de controle são comumente definidos em Prolog de forma recursiva. Por exemplo, o predicado **acrescente**, que adiciona ao final de uma lista [X|L1] uma lista L2, gerando uma nova lista [X|L3], pode ser definido da seguinte forma:

```
acrescente ([], L, L).
acrescente ([X|L1], L2, [X|L3]) :- acrescente (L1, L2, L3).
```

As cláusulas de um programa em Prolog podem ser interpretadas de forma **declarativa** ou **procedural**. No primeiro caso, tais cláusulas definem objetos e relações entre objetos, podendo o programa ser interpretado como uma **linguagem de especificação** [Davis 85]. No exemplo acima, a primeira cláusula informa que acrescentar a uma lista vazia uma lista resulta na própria lista. A segunda, entre outras informações, especifica que será acrescentada a uma lista não vazia uma segunda lista, gerando uma terceira com a cabeça da primeira, se à cauda da primeira for acrescentada a segunda, gerando a cauda da terceira.

Na semântica procedural define-se como o programa é executado. As cláusulas podem ser vistas como uma base de dados e são executadas como funções, quando uma submeta a ser satisfeita é escolhida. No exemplo acima, o conjunto de cláusulas pode ser usado pelo menos de dois modos:

(a) como **reconhecedor** de relações. Por exemplo, se a seguinte meta é escolhida

```
?- acrescente ([a], [b,c], [a,b,c]).
```

o resultado da execução será **sucesso**;

(b) como **gerador** de respostas que satisfazem a uma determinada relação. Por exemplo, se a seguinte meta é escolhida

```
?- acrescente (Y, Z, [a]).
```

a primeira resposta será Y = [] e Z = [a]. Caso uma nova alternativa seja solicitada, a segunda resposta será Y = [a] e Z = [].

Estruturas de controle podem também ser especificadas em Prolog, utilizando-se recursividade. Por exemplo, um **laço** pode ser definido da seguinte forma:

```
execute (Inicio, Fim) :- predicado (Inicio, Seguinte),
                         execute (Seguinte, Fim).

execute (Inicio, Fim) :- predicado (Inicio, Fim).
```

a primeira cláusula **execute** gera o laço, enquanto a segunda permite parar a execução desse laço.

3. Especificação de serviços e protocolos em Prolog

O desenvolvimento de serviços e protocolos de comunicação é normalmente realizado através de uma sequência de fases, iniciando-se com a formulação de intenções (**especificação informal**) e convergindo para uma **implementação**. Uma das fases intermediárias é a **especificação formal**, que tem por objetivo a descrição clara e concisa de serviços e protocolos. As **especificações de serviços** descrevem os serviços fornecidos pelos níveis hierárquicos de protocolos e as **especificações de protocolos** descrevem o comportamento das entidades a serem implementadas.

Máquinas de estado finito (MEF) são frequentemente utilizadas como forma de representação do comportamento de serviços e protocolos. Os elementos constituintes de uma MEF são **estados** (estados principais) e **interações**. Uma MEF ao receber uma interação de **entrada** (proveniente de seu ambiente) ou **espontânea** (interna) executa uma **transição**. Ao executar uma transição a MEF mudará (ou não) de estado, assim como produzirá (ou não) uma interação de **saída** para o seu ambiente.

A fim de permitir a introdução dos **parâmetros** presentes nas interações (**primitivas de serviço ou unidades de dados de protocolo**) e a fim de introduzir **variáveis de estado adicionais**, as MEF são estendidas (**MEFE**). Isso permite a descrição completa das transições, que se contituem no corpo das especificações formais de serviços e protocolos, representados através de MEFE. Um exemplo de linguagem de especifição baseada em MEFE é apresentado em [Lopes 85].

O uso de programação lógica e Prolog para a especificação de serviços e protocolos é proposto em [LoSiUr 84] e [Sidhu 83] respectivamente. Ambas as especificações, embora realizadas com objetivos diferentes e a partir de modelos diferentes, contêm duas partes distintas: uma **lógica** e outra de **controle**. No componente lógico dessas especificações a forma utilizada para a descrição das transições é semelhante a:

```
trans (Estado_atual, Estado_seguinte, Entrada, Saida)
```

onde as variáveis **Estado_atual** e **Estado_seguinte** representam os estados principais antes e depois da transição respectivamente; **Entrada** e **Saida** representam as interações (sem os seus parâmetros).

A fim de permitir a inclusão das variáveis de estado adicionais e dos parâmetros das interações, nós propusemos o seguinte modelo em [BoDsLoSaUr 85]:

```
trans (Estado_atual, Estado_seguinte, Entrada, Saida) :- condicao, acao.
```

onde os argumentos da **cláusula trans** são listas. Os dois primeiros são

constituidos de um estado principal seguido de variáveis de estado adicionais e os dois últimos são constituidos de uma interação seguida de seus parâmetros. Por exemplo:

```
Estado_atual = [fechado, Em_uso, Opcoes, .....]

Entrada = [t_CONreq, Para_endereco, Opcoes_propostas, .....]
```

O corpo da cláusula **trans** é constituido de dois conjuntos de predicados, isto é, **condicao** e **acao**. O primeiro habilita a transição e o segundo atualiza algumas variáveis de estado (as demais são atualizadas diretamente em Estado_seguinte).

Para a especificação das transições de um módulo composto de um conjunto de submódulos, por exemplo, duas entidades de protocolo e um módulo de serviço (Fig.1), a estrutura das variáveis, que representam os estados atual e seguinte desse sistema, pode ser do tipo:

```
[Estado_entidade1, Estado_entidade2, Estado_servico]
```

A estrutura das variáveis, que representam as interações de entrada e de saída, pode ser do tipo:

```
[Interacao_servico1, Interacao_servico2]
```

sendo que as interações ocorrem nos **pontos de acesso ao serviço_N**, isto é, PASN1 e PASN2 (Fig.1).

Uma vez que cada submódulo do sistema suporta duas entradas e duas saídas, o modelo de transição apresentado (**cláusula trans**) deve ser ligeiramente modificado para:

```
t (Estado_atual, Estado_seguinte, [Entrada1, Entrada2], [Saida1, Saida2])
:-
condicao, acao.
```

onde a ausência de uma entrada ou saída é representada por [].

As **transições globais** do módulo apresentado na Fig.1, que fornece o serviço_N e onde as interações são do tipo "rendez-vous" (isto é, entrada e saída simultâneas nas interfaces), podem ser descritas através do modelo:

```
tSN ([E1a, E2a, ESNIa], [E1s, E2s, ESNIs], [ESN1, ESN2], [SSN1, SSN2])
:-
t (E1a, E1s, [ESN1, SSNI1], [SSN1, ESNI1]),
t (E2a, E2s, [ESN2, SSNI2], [SSN2, ESNI2]),
tSNI (ESNIa, ESNIs, [ESNI1, ESNI2], [SSNI1, SSNI2]).
```

Se um modelo com **filas de interação** for utilizado (por exemplo, filas entre as entidades e o submódulo de serviço da Fig.1), às variáveis de estado, presentes nos argumentos da cláusula **tSN**, devem ser acrescentadas variáveis representando os estados das filas. Ao corpo da cláusula **tSN** devem ser introduzidos predicados **acrescente** e **retire** (semelhantes ao definido na seção 2), que serão responsáveis pelas operações a serem realizadas nessas filas e pelas atualizações de seus estados.

No modelo de transição global proposto, uma transição do módulo consiste de transições simultâneas de seus submódulos. Caso filas de interação sejam utilizadas, é possível definir-se um outro modelo, onde uma transição de um dos submódulos acarreta numa transição global do sistema. Nesse caso, a cláusula tSN deverá ser constituida de um conjunto de cláusulas alternativas, uma para cada submódulo [UrPr 84], ou uma para cada tipo de transição dos submódulos [Sidhu 83].

4. Utilização de Prolog para a validação de serviços e protocolos

O objetivo da validação é verificar se representações de um mesmo sistema, apresentadas em diferentes níveis de abstração, são conformes. As atividades de validação são normalmente divididas em duas categorias: verificação e teste.

Verificação procura demonstrar logicamente propriedades dos serviços e protocolos, tais como:

(a) ausência de impasses ("deadlock freeness"): o sistema nunca atingirá um estado, a partir do qual não será possível uma nova transição;

(b) ausência de ciclos improdutivos ("livelock or tempo-blocking freeness"): não ocorrerão ciclos improdutivos entre qualquer subconjunto de estados do sistema;

(c) finalização ("termination"): a partir de um estado, existirá sempre um "caminho" através dos estados, permitindo ao sistema atingir um estado final previsto na especificação.

Teste procura demonstrar propriedades semelhantes, mas através da execução controlada do sistema e comparando o comportamento observado em relação ao especificado.

Especificações de serviços e protocolos baseadas em MEFE podem ser verificadas, em relação às propriedades acima descritas, através da geração de caminhos de interação pertencentes ao espaço de estados do sistema representado. Em [Sidh 83], o componente de controle da especificação do sistema é constituido de um conjunto de cláusulas, que permitem verificar se existe um caminho de interação entre um estado inicial e um estado meta desejado. Em [LoSiUr 84], o componente de controle explora as propriedades de reconhecimento e geração da linguagem Prolog (descritas na seção 2), para reconhecer se uma sequência de interações é válida, ou para gerar todas as possíveis sequências válidas de interação a partir de um determinado estado do sistema.

Sequências de teste para serviços e protocolos são apresentadas em [UrPr 83]. Tais sequências são especificadas numa gramática livre de contexto atribuida (para permitir a inclusão dos parâmetros das interações e alguns parâmetros de controle). A gramática obtida é então implementada em Prolog e a implementação é executada, de forma controlada, a fim de gerar as sequências de teste. Entretanto, essa especificacção, por não conter a noção de estado global do sistema, pode ser usada somente como geradora e não como reconhecedora de sequências válidas de interação.

Os artigos acima citados procuram demonstrar as vantagens de

dispor-se de especificações executáveis de sistemas e em particular as vantagens propiciadas pela programação lógica e Prolog, ao permitirem a divisão de uma especificação numa parte lógica e outra de controle. Por exemplo, a execução de uma sequência de transições pode ser descrita através de uma estrutura de controle, semelhante ao laço apresentado na seção 2, da seguinte forma:

```
execute (Estado_inicial, Estado_final,
        [Entrada|Entrada_cauda], [Saida|Saida_cauda])
:-
trans (Estado_inicial, Estado_seguinte, Entrada, Saida),
execute (Estado_seguinte, Estado_final, Entrada_cauda, Saida_cauda).

execute (Estado_inicial, Estado_final, [Entrada], [Saida])
:-
trans (Estado_inicial, Estado_final, Entrada, Saida).
```

A primeira cláusula indica que um **caminho de interação**, partindo de um **Estado_inicial** e terminando num **Estado_final**, é possível, se:

(a) existe um predicado **trans**, cujas **Entrada** e **Saida** correspondem respectivamente ao primeiro elemento da sequência de entrada e ao primeiro elemento da sequência de saída, que partindo de um **Estado_inicial** leva a um **Estado_seguinte** e

(b) existe um possível **caminho de interação remanescente**, cujas sequências de entrada e saída correspondem respectivamente às caudas das sequências anteriores, que partindo do **Estado_seguinte** leva ao **Estado_final**.

Esse componente de controle pode ser usado para:

(a) **simular execução**: a partir de um estado inicial e de uma dada sequência de interações de entrada, determina-se os possíveis estados finais com suas respectivas sequências de interações de saída;

(b) **verificar um traço**: a partir de um estado inicial e de um traço observado (sequência de entradas e saídas) de uma implementação, determina-se se o traço é válido em relação à especificação e determina-se os possíveis estados finais da implementação, após a execução das transições correspondentes a esse traço;

(c) **encontrar caminhos de interação**: a partir dos estados inicial e final, determina-se as possíveis sequências de entrada e saída correspondentes às possíveis sequências de transições, que levam do estado inicial ao estado final.

Variações dessa estrutura de controle, que permitem, por exemplo, a definição de uma **história de execução** (sequência de elementos da forma [Estado, Entrada, Saida]), são apresentadas em [BoDsLoSaUr 85].

Um sistema, que pode utilizar Prolog como ferramenta para testes de serviços e protocolos de comunicação, é descrito em [LoDsBo 85]. Tal sistema é constituido basicamente de três módulos: o primeiro especificando uma referência para os testes, o segundo é um gerador de sequências de teste e o último um reconhecedor de sequências de interações provenientes da implementação sob teste. Em Prolog, esse sistema poderia corresponder à definição de um componente lógico, obtido a partir da

especificação do serviço ou protocolo, e de dois (ou um somente) componentes de controle responsáveis pela geração e reconhecimento das sequências de interações.

5. Conclusão

Nesse trabalho, uma análise de algumas possibilidades de utilização de Prolog na área de Protocolos de Comunicação é realizada. Um modelo em Prolog para as transições de sistemas, cujas especificações são baseadas em Máquina de Estado Finito Estendida, é proposto. Em seguida, um exemplo de uma estrutura de controle, que permite explorar (para fins de validação) a descrição das transições de um módulo, é apresentado.

As grandes vantagens de especificações escritas em Prolog parecem residir no fato de serem executáveis e de serem reversíveis, isto é, é possível definir-se um componente lógico e outro de controle, que permitam a geração e reconhecimento de sequências de interações, facilitando assim as tarefas de verificação e teste de serviços e protocolos de comunicação.

Como proposta para trabalhos futuros, sugerimos a especificação de um exemplo completo, utilizando os modelos em Prolog apresentados nesse artigo, e submeter esse exemplo a uma série de métodos de validação. Isto poderá permitir uma melhor visualização das condições de aceitabilidade de Prolog, como linguagem de especificação de serviços e protocolos, sobretudo no que tange às questões de desempenho dessa linguagem.

6. Referências

[BoDsLoSaUr 85] : G.V. Bochmann, R. Dssouli, W. Lopes de Souza, B. Sarikaya, H. Ural, "Use of Prolog for Building Protocol Design Tools", publicação interna No 524 (submetida ao Fifth Int. Workshop on Protocol Specification Verification and Testing), Univ. de Montréal, março 1985.

[ClMe 81] : W.F. Clocksin, C.S. Mellish, "Programming in Prolog", Springer- Verlag, 1981.

[Davis 85] : R. Davis, "Executable Specifications as Basis of Program Life Cycle", anais do 18th Hawaii Int. Conf. on System Sciences, vol II, pp. 722-733, jan 1985.

[Kowa 79] : R. Kowalski, "Logic for Problem Solving", Elsevier North-Holland, 1979.

[LoDsBo 85] : W. Lopes de Souza, R. Dssouli, G.V. Bochmann, "Ambiente de Teste para Protocolos de Comunicação", anais do 3 Simpósio Brasileiro de Redes de Computadores", abril 1985.

[Lopes 85] : W. Lopes de Souza, "Utilização dos Conceitos de Módulo, Porta e Canal em Especificações de Serviços, Protocolos e Interfaces de Comunicação", anais do 3 Simpósio Brasileiro de Redes de Computadores, abril 1985.

[LoSiUr 84] : L. Logrippo, D. Simon, H. Ural, "Executable Description

of the Transport Service in Prolog", anais do Fourth Int. Workshop on Protocol Specification Testing and Verification, North-Holland, 1984.

[Sidhu 83] : D.P. Sidhu, "Protocol Verification via Executable Logic Specifications", Protocol Specification, Testing and Verification, III, North-Holland, pp. 237-248, 1983.

[UrPr 83] : H. Ural, R.L. Probert, "User-guided Test Sequence Generation", Protocol Specification, Testing and Verification, III, North-Holland, pp. 421-436, 1983.

[UrPr 84] : H. Ural, R.L. Probert, "Testing Specifications and Designs of Communication Protocols and Services", publicacao interna TR-84-18, University of Ottawa , 1984.

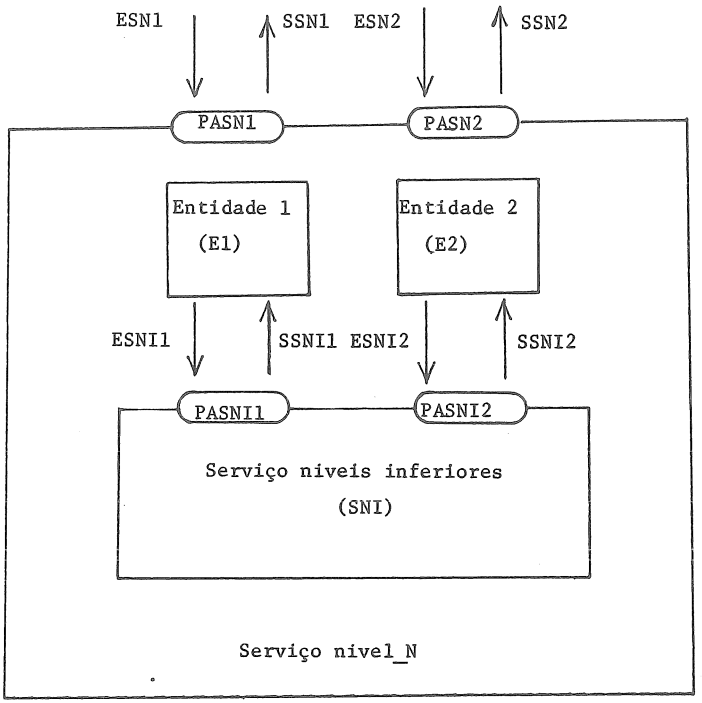


Fig. 1. Sistema composto fornecendo o serviço do nivel_N

TÉCNICAS DE MULTIPLEXAÇÃO POR DIVISÃO NO TEMPO PARA INTERCONEXÃO PROCESSADOR MEMÓRIA EM SISTEMAS DE MULTIPROCESSADORES: UM ESTUDO COMPARATIVO

MARCOS A.G. BRASILEIRO* E J. ANTÃO B. MOURA*

SUMÁRIO

Quatro técnicas de multiplexação por divisão no tempo são investigadas para determinar a mais adequada para a comunicação processador-memória em sistemas com multiprocessadores. O estudo é realizado através de simulações.

ABSTRACT

Four TDM scanning disciplines are simulated in order to determine the technique best suited for the interconnection mecanism of a multiprocessor system.

* DEE/CCT/UFPb
Grupo de Redes de Computadores
Fone: (083)322.1948
58.100 — Campina Grande - Pb.

1. INTRODUÇÃO

A proliferação de micro e mini computadores tem instigado o projeto e implementação de sistemas com multiprocessadores para explorar o paralelismo inerente das tarefas submetidas. O objetivo destas implementações é geralmente, melhorar a vazão de tarefas processadas. Para tanto, vários processadores operam simultaneamente em várias sub-tarefas ou em um número de tarefas. É importante notar que os projetos desses sistemas a multiprocessadas são realizáveis com o nível de tecnologia de hardware já atingido; o desempenho dos sistemas porém, ainda é distante do caso ideal. Em geral, sabe-se que qualquer arquitetura de computador sofre uma degradação em desempenho devido às limitações impostas pelo mecanismo de interconexão, responsável por interligar as suas várias unidades funcionais, i.e., processadores, unidade de entrada e saída (E/S) e blocos de memória.

O sistema com multiprocessador de interesse aqui é a máquina de Instruções Múltiplas e fluxos de Dados Múltiplos (IMDM) [FLYN 72, HAND 77]. O mecanismo de interconexão empregado num sistema IMDM oferece vários graus de "acoplamento" entre as unidades funcionais. Assim, as redes de computadores ou processadores distribuídos podem ser classificadas como sistemas IMDM acoplados frouxamente ("loosely coupled"). Nosso interesse, entretanto, focaliza as organizações acopladas "rigidamente" ("tightly coupled") onde os processadores e memórias são conectados via alguma malha de interconexão, constituindo-se em sistemas fisicamente "fechados" ou restritos a unidades isoladas, com identidade própria. Como esperado, a malha de interconexão pode ser um fator limitante com respeito à vazão.

O problema de interconectar sub-unidades tem recebido atenção considerável, destacando-se as redes telefônicas reconfiguráveis [BATC 68], os arranjos celulares de Kautz [KAUT 68], redes de permutação serial [JOEL 68], redes de Lawrie [LAWR 73], interconexão "shuffle" [STON 75], memória multiporta [ENSL 77]. Sugestões de interconexão via um único barramento com acesso aleatório (como em redes locais) foram também feitas [MEHR 79]. Entre estes, os mecanismos de interconexão mais freqüentemente adotados em sistemas IMDM são o arranjo "cross point", o barramento compartilhado no tempo, memórias multiportas e redes de chaves binárias. Apesar de que o escopo dos esquemas de interconexão é bem mais amplo e complexo, estas categorias contudo, formam uma base útil para a discussão da organização de máquinas IMDM.

Enslow [ENSL 77] e Pearce [PEAR 77] investigaram e compararam o desempenho dos mecanismos mais freqüentes e concluíram que o sistema de interconexão mais simples, com menor custo para um ambiente de multiprocessadores, é um barramento compartilhado no tempo, o qual conecta todas as unidades funcionais. Uma maneira de se efetuar este compartilhamento, conhecida por Multiplexação por Divisão no Tempo (MDT), é alocar um intervalo de tempo (chamado de quadro ou slot) a cada unidade funcional, a qual acessa o barramento apenas durante o quadro a ela alocado. Um controlador (unidade de controle de multiplexação) é responsável por alocar os quadros às unidades funcionais. Uma alternativa a esta maneira é fazer com que cada unidade que deseje utilizar o barramento determine a disponibilidade do mesmo e o acesso num intervalo conveniente, minimizando a chance de possí

veis conflitos. O controle de acesso assim especificado é dito ser distribuído. A vantagem do controle centralizado é o seu baixo custo.

O objetivo deste artigo é investigar a adequação de algumas técnicas de MDT sob controle centralizado para acesso a uma configuração de barramento comum, no contexto de sistemas IMDM.

2. ARQUITETURA IMDM DE INTERESSE

Para efeito do presente estudo, considera-se um cenário para a comunicação entre as unidades funcionais (processadores e bancos de memória) de um sistema IMDM que corresponde à comunicação unidirecional oriunda de N Processadores para M Bancos de Memória. Este cenário é de interesse pois, enquanto simples, permite considerações preliminares sobre as características da arquitetura IMDM com interconexão via métodos MDT. Ele será também utilizado numa avaliação comparativa destes métodos a fim de selecionar o método mais adequado para uso em sistemas IMDM.

2.1. Interconexão das Unidades Funcionais

O cenário em consideração é baseado na arquitetura de um sistema IMDM, ilustrado na figura 1. Como mostra a figura, as unidades funcionais são "rigidamente" acopladas ("tightly coupled") através de um barramento. Os N Processadores compartilham o barramento através de um multiprocessador com N linhas de entrada (uma para cada processador) e com o barramento na sua saída. O Decodificador é utilizado para fazer o encaminhamento das mensagens no barramento a um dos M Bancos de Memória. Para tanto, cada mensagem no barramento contém o endereço do Banco de Memória destinatário.

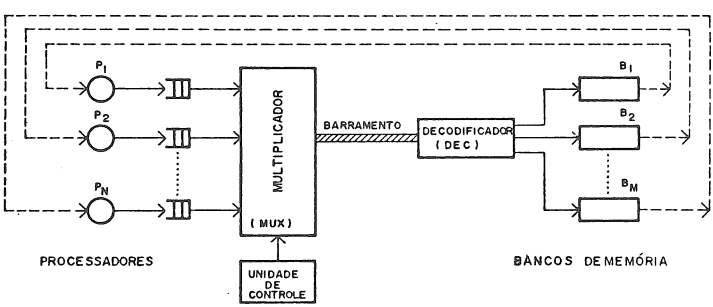


Figura 1: Conexão Processador-Memória

Convém observar que o barramento na figura 1 é compartilhado (por MDT) entre as unidades que a ele se ligam. MDT é um método de multiplexação no qual o tempo de um canal multiplexado é dividido em quadros (de duração fixa ou variável). Os quadros são então alocados, em instantes diferentes, a diferentes usuários do canal. A alocação pode ser repetida regularmente (MDT Síncrona, MDTS) ou pode ser feita sob demanda (MDT Assíncrona ou MDTA) [DAVI 73].

Com métodos MDT, apenas uma unidade pode ter acesso ao barramento por vez. Assim, para que não se percam mensagens geradas por uma unidade enquanto ela não tem acesso ao barramento e/ou para liberar mais rapidamente os processadores de tarefas de comunicação (ex.: evitar que um processador fique à espera do uso do barramento), fornecem-se buffers a cada linha de entrada dos multiplexadores (vide figura 1). As mensagens são então armazenadas nos buffers enquanto aguardam transmissão no barramento.

2.2. Disciplina de Atendimento

Quando várias unidades funcionais compartilham um barramento por meio de métodos MDT, deve-se definir uma "disciplina de atendimento" (implementada pelo controlador do multiplexador) para que cada unidade possa requisitar e obter controle do barramento (a fim de efetuar suas transmissões). A dificuldade maior quanto à disciplina de atendimento diz respeito à resolução de conflitos nos pedidos das unidades para acesso ao barramento. Como vimos, apenas uma unidade deve ter acesso em qualquer instante. O presente artigo trata de quatro disciplinas de atendimento, as quais são descritas abaixo.

2.2.1. Disciplina de Atendimento Cíclico (MDTS)

A disciplina de atendimento MDTS é cíclica no sentido de que o controlador do multiplexador reserva um quadro fixo para cada linha de entrada independentemente de sua necessidade de transmissão. Este comportamento implica em que as linhas sejam atendidas em seqüência, numa freqüência fixa. Com a disciplina cíclica, a seqüência de quadros no tempo é como mostra a figura 2.

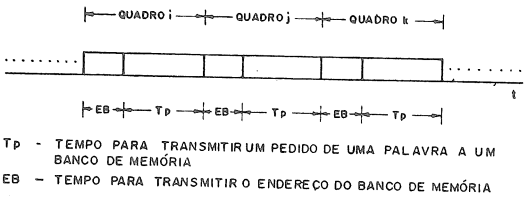


Figura 2: Seqüência de Quadros em MDTS

2.2.2. Disciplina com Lista de Chamada (LIC)

Um controlador implementando LIC verifica o estado do buffer de cada linha. Se o buffer sendo verificado tem algo a transmitir, o controlador aloca um quadro ao primeiro item no buffer. Caso contrário, o controlador passa para a próxima linha segundo uma "ordem de chamada" fixa. Observe que a chamada segue um procedimento síncrono já que um intervalo de tempo fixo é gasto para verificar cada buffer, independentemente do seu estado. Esta disciplina apresenta uma seqüência de quadros como ilustra a figura 3.

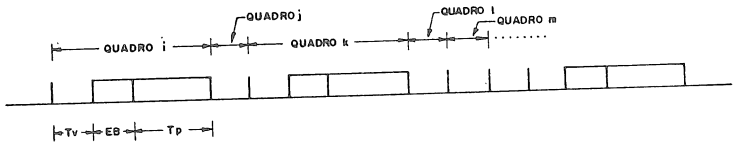


Figura 3: Possível Seqüência de Quadros com LIC

2.2.3. Disciplina com Verificação Múltiplas (VEM)

VEM é semelhante a LIC, mas com a diferença de que, se o buffer se encontrar vazio inicialmente, o controlador insiste na verificação por um certo número de vezes, n, antes de passar à próxima linha. Por exemplo, se n=3, o controlador desperdiçará, no máximo $3T_t$ segundos antes de desistir e passar para a linha seguinte. A figura 4 ilustra uma possível seqüência de quadros com a disciplina VEM para o caso de n=3.

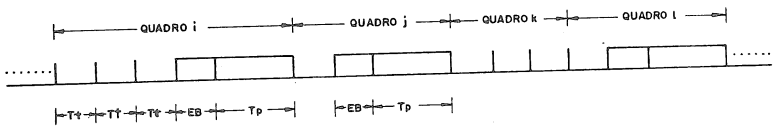


Figura 4: Exemplo de uma Seqüência de Quadros com VEM

2.2.4. Disciplina com Verificação Aleatória (VEA)

VEA faz com que a verificação de um dado buffer de linha se dê em um certo período de tempo, s, de acordo com uma probabilidade p(s). Uma possibilidade é fazer p(s) uma função do número de linhas de entrada do multiplexador, como por exemplo (vide figura 1),

$$p(s) = \frac{1}{N}$$

A duração da verificação, como no caso da disciplina VEM, é T_t seg. A seqüência de quadros com VEA é semelhante a obtida com LIC (figura 3). Na próxima seção, investigamos as características de desempenho das quatro disciplinas de atendimento apresentadas acima, sob as condições do cenário na figura 1.

3. AVALIAÇÃO DO DESEMPENHO DAS DISCIPLINAS DE ATENDIMENTO

3.1. Descrição do Modelo

A definição do modelo para o sistema IMDM da figura 1 requer suposições e considerações sobre os aspectos operacionais do sistema. Primeiro, assumimos que durante um intervalo de tempo Ts, cada processador, independentemente dos outros, gera um pedido para um dos Bancos de Memória apenas uma vez, com probabilidade λ. Segundo, os tempos de interchega

das de pedidos tem distribuição uniforme com média Ts. Assim, quando um processador faz um pedido, apenas após um intervalo médio de tempo de Ts segundos é que ele fará um outro pedido com probabilidade λ. Esta suposição leva então a uma distribuição binomial para o número de pedidos durante o intervalo Ts [RUDI 71]. Deve-se ressaltar que a distribuição de Poisson para o número de pedidos, freqüentemente adotada a fim de facilitar a solução do modelo [BIRD 62, DOR 67, CHU 69], é aqui descartada devido às características reais do sistema - tais como número finito de processadores, buffers de linha com capacidade finita, etc. Quando um processador pede uma palavra (uma instrução ou um dado) à memória, o processador gera o endereço da palavra e coloca o pedido no seu buffer de linha. Os pedidos de palavra são armazenados no buffer de linha por ordem de chegada. Assume-se que o cumprimento dos pedidos de palavra é fixo e composto apenas do endereço da palavra, "EP", e do endereço do Banco de Memória, "EB".

Cada pedido de palavra permanece no buffer de linha até que o multiplexador, controlado pela disciplina de atendimento, remova o pedido e o enderece ao banco de memória via o barramento. Como o sistema IMDM sendo discutido é "rigidamente acoplado", o barramento tem uma alta capacidade de transmissão. Assumimos que os atrasos de transmissão dos pedidos de palavra no barramento são desprezíveis. Uma última suposição é a de que a decodificação dos endereços da palavra leva Td segundos. O decodificador tem condições de detectar se o banco de memória endereçado está ocupado ou não. Caso o esteja, o pedido é bloqueado, permanecendo na cabeça do buffer de linha aguardando o próximo acesso ao barramento. Caso contrário, o acesso é garantido depois de um atraso de Ta segundos, o equivalente ao tempo de acesso à memória.

3.2. Medidas de Desempenho

As medidas de desempenho de interesse são:

L - Comprimento Médio das filas de pedidos de palavra nos buffers de linha.

E - Tempo Médio de Espera dos pedidos de palavra nos buffers de linha. "E" é a média dos intervalos de tempo definidos entre a chegada de um pedido de palavra ao buffer e o início de sua transmissão através do multiplexador.

P - Probabilidade de Bloqueio de pedidos de palavra nos bancos de memória.

V - Vazão Média do sistema IMDM. "V" fornece o número de pedidos (por unidade de tempo) que passam pelo multiplexador e obtêm sucesso no acesso ao banco de memória endereçado.

A solução do modelo descrito na seção 3.1 mostrará o comportamento das medidas acima em função de variações nos parâmetros N, M e do sistema IMDM com cada uma das quatro disciplinas de atendimento de interesse.

3.3. Solução do Modelo

As suposições feitas na seção 3.1 e a dependência estatística entre as filas que se for

mam nos buffers de linha tornam intratável a solução analítica exata do modelo. Assim, decidimos por uma simulação do sistema.

Um simulador gatilhado por eventos foi implementado na linguagem C e executado num PDP 11 com sistema operacional UNIX*. O simulador inclui todas as características julgadas importantes e inerentes a cada disciplina de atendimento em conjunção com a arquitetura e ambiente da figura 1.

O programa simulador foi dividido em seções modelando os processadores, os buffers de linha, multiplexador, decodificador e bancos de memória. Foram implementadas ainda as seguintes rotinas de suporte: "escalador de eventos" e "coletor de estatística". As semáforas permitidas de Dijkstra (P e V) foram também implementadas. Cada parte do programa foi escrita como co-rotina ou procedimento. Maiores detalhes acerca do programa simulador são fornecidos em [BRAS 85].

3.4. Resultados

O simulador da seção anterior é utilizado para fornecer as características de desempenho das quatro disciplinas de atendimento da seção 2.2, a saber: MDTS, LIC, VEM e VEA.

A tabela 1 fornece os valores dos parâmetros utilizados nas execuções do simulador. Estes valores foram escolhidos para aproximar atrasos típicos [TTL 76] das unidades funcionais. Observe-se, que o simulador é genérico e qualquer outro conjunto de valores poderia ter sido utilizado. Os resultados abaixo têm intervalos de confiança variando de ±0,12% a ±5,23% dos valores verdadeiros com um nível de confiança de 95%.

Parâmetro	Valor
uts - unidade de tempo simulado	25 ns
Ts - média dos tempos de interchegadas	64 uts
Tm - tempo de transporte de um pedido através dos multiplexadores	3 uts
Tv - tempo de verificação de um buffer de linha	1 uts
Td - tempo de decodificação do endereço do banco de memória	3 uts
Ta - tempo de acesso à memória	13 uts
N - número de processadores	2, 4, 8 e 16
M - número de bancos de memória	2, 4, 8 e 16

Tabela 1: Valores dos Parâmetros utilizados no modelo de Simulação do Sistema IMDM da figura 1.

* UNIX é marca registrada da AT&T.

As figuras 5, 6 e 7 apresentam, respectivamente, resultados para L, E e P em função de λ, a probabilidade de um processador gerar um pedido de palavra em Ts segundos, quando N, o número de processadores, é fixado em 8 e M = 2, 4, 8 (M, é o número de bancos de memória).

É importante observar que nas figuras 5, 6 e 7, os pedidos bloqueados nos bancos de memória são colocados de volta na cabeça dos respectivos buffers de linha. O desempenho de cada disciplina de atendimento apenas como função da atividade de geração pura de pedidos, sem re-inserção de pedidos bloqueados nos buffers de linha, é ilustrado nas figuras 8, 9, 10 e 11. Nestas figuras, adotou-se λ=1.0 para se observar o comportamento das disciplinas sob condições de alta intensidade de tráfego.

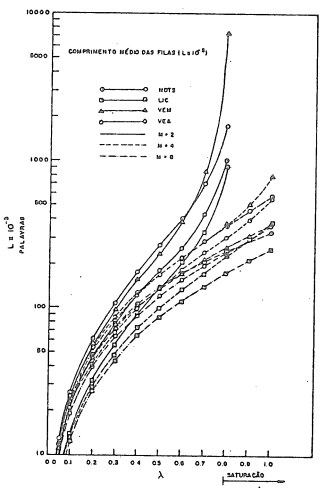


Figura 5: L vs λ para as disciplinas MDTS, LIC, VEM e VEA no caso de Comunicação Processador-Memória.

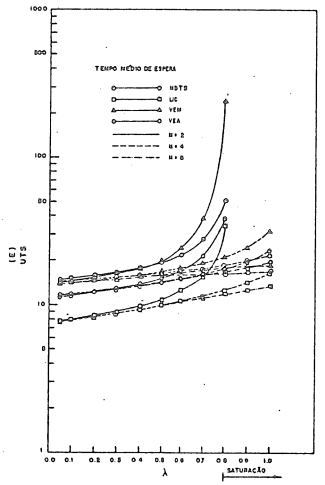


Figura 6: E vs λ para as disciplinas MDTS, LIC, VEM e VEA no caso de Comunicação Processador-Memória.

As figuras 5 e 6 mostram que a disciplina LIC oferece o melhor desempenho para L e E. Com relação a P, VEM e VEA apresentam menor probabilidade de bloqueio, mas à medida que λ e M crescem, as diferenças se tornam insignificantes.

No caso das figuras 8-11, LIC e MDTS são em geral, superiores. Pode-se observar que LIC é melhor que MDTS com relação a L, E e V quando $M \leq 11$. A partir daí, MDTS se torna mais vantajosa. Isto porque o tráfego, sendo mais intenso, não justifica a verificação de cada buffer de linha (feito por LIC) já que praticamente todos os buffers terão pedidos para transmitir. Nesta situação, apenas poucos quadros da disciplina MDTS serão desperdiçados. Estas considerações são novamente aplicáveis às curvas da vazão na figura 10.

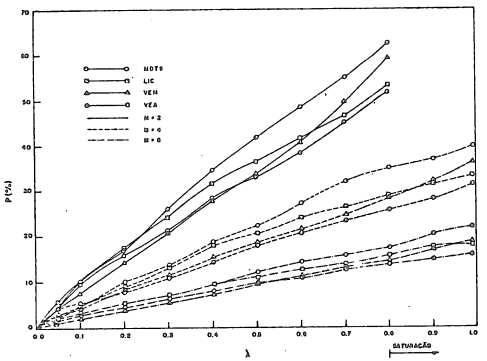


Figura 7: P vs λ para as disciplinas MDTS, LIC, VEM e VEA no caso de Comunicação Processador-Memória.

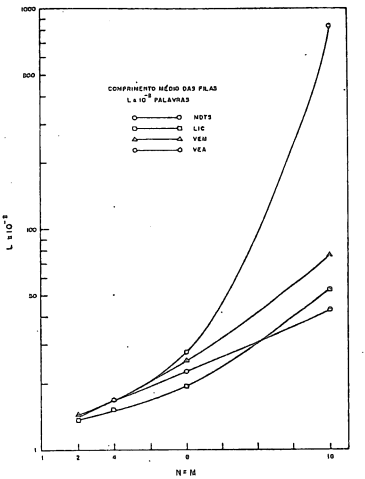


Figura 8: L vs (N=M) para as disciplinas MDTS, LIC, VEM e VEA no caso de Comunicação Processador-Memória.

A disciplina VEA apresenta a pior vazão. Isto se deve à aleatoriedade com que VEA serve cada uma das linhas do multiplexador. De fato, como ela serve um determinado buffer de linha com uma probabilidade igual a 1/N, pode acontecer que algum buffer nunca seja servido (o que não ocorre com as outras três disciplinas). Quando N cresce, as chances de serviço diminuem fazendo com que VEA aparesente uma redução acentuada na vazão.

Finalmente, com relação a P vs (N=M) e λ=1, o melhor desempenho é alcançado com a disciplina VEA. Como o acesso dos processadores aos bancos de memória é aleatório, é possível que um mesmo buffer de linha seja atendido mais de uma vez consecutivamente, reduzindo assim a possibilidade de bloqueio para esse buffer.

A probabilidade de K verificações sucessivas de um mesmo buffer de linha, com a disciplina VEA, é $(1/N)^K$. Para valores de N grande, essa probabilidade é pequena. Assim, comparando-se as curvas para P das disciplinas VEA e LIC na figura 11 vemos que as diferenças são, no máximo, 5% e que as curvas tendem a se confundir quando N cresce. VEA perde pois sua vantagem sobre LIC. Convém notar também que a complementação de LIC, comparada à implementação de VEA, é mais simples. No cômputo global-figuras 5-11 - LIC é a disciplina com melhor nível de desempenho.

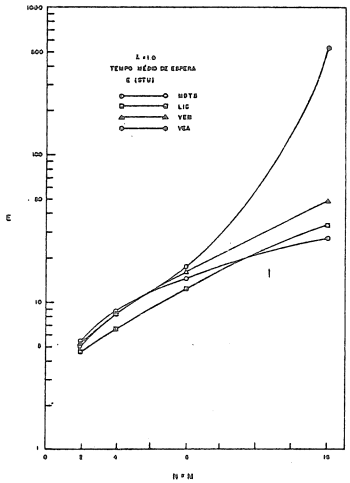


Figura 9: E vs (N=M) para as disciplinas MDTS, LIC, VEM e VEA no caso de Comunicação Processador-Memória

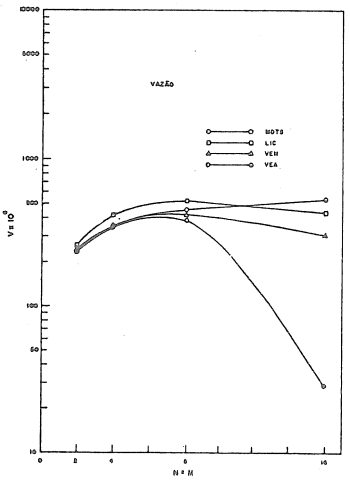


Figura 10: V vs (N=M) para as disciplinas MDTS, LIC, VEM e VEA no caso de Comunicação Processador-Memória

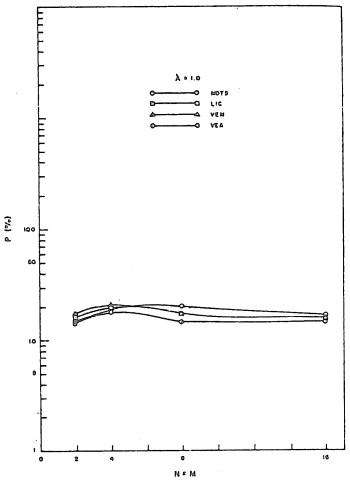


Figura 11: P vs (N=M) para as disciplinas MDTS, VEM e VEA no caso de Comunicação Processador-Memória.

4. CONCLUSÕES E SUGESTÕES

No projeto de um sistema com multiprocessadores (IMDM), uma questão crítica é prover um método de comunicação eficiente e de baixo custo entre os processadores e os bancos de memória. Técnicas de Multiplexação por Divisão no Tempo (TDM) propiciam esta comunicação com as qualidades desejadas. A vazão do sistema porém, depende da disciplina de atendimento. Este artigo apresentou um estudo comparativo de quatro disciplinas: Atendimento Cíclico (MDTS), Lista de Chamadas (LIC), Verificação Múltipla (VEM) e Verificação Aleatória (VEA). Inicialmente, apenas quando a comunicação dos processadores para as memórias foi considerada, chegou-se às seguintes observações:

1) As quatro disciplinas apresentam desempenho comparável quando a atividade dos processadores é baixa.

2) LIC aparenta ser a melhor disciplina para pequenos sistemas. Para um grande número de processadores, porém, MDTS é superior.

3) Os experimentos de simulação mostram que o tempo de acesso à memória, Ta, é um fator importante. Reduzindo-se Ta, minimiza-se a probabilidade de bloqueio, P, e aumenta-se a vazão.

4) Observa-se também que P diminui quando um maior número de memórias, M, é empregado. O custo de tal sistema é maior, obviamente.

Limitando a investigação a sistemas multiprocessadores pequenos, LIC deve ser estudada em maiores detalhes sob um cenário de comunicação em ambas direções, i.e., processador-memória-processador. Este cenário é mais realista (e por isto mais complexo), exigindo um modelo mais refinado. O seu estudo não foi apresentado aqui por questões de brevidade, mas, o leitor interessado pode dirigir-se a [BRAS 85] para mais informações.

Como sugestões para continuação deste trabalho, destacam-se um estudo mais detalhado dos sistemas IMDM-LIC com memória locais juntamente com o mecanismo de Busca Prévia ("Prefetching") a fim de estabelecer a sua utilidade e limitações; e uma análise matemática, possivelmente através da Teoria das Filas, do referido sistema que proporcione maior eficiência de solução, haja visto que o custo de execução do simulador aqui utilizado é apreciável. Referências para essas sugestões seriam [BUZE 73, PEAR 77, WONG 78]. Finalmente, os autores agradecem ao CNPq pelo suporte a este trabalho.

REFERÊNCIAS

[BATC 68] Batcher, K.E. - "Sorting Networks and their Applications" - Proc. do AFIPS, SJCC, 1968, pp. 307-314.

[BIRD 62] Birdsall, T.G., Ristenbatt, M.P. e Weinstein, S.B. - "Analysis of Asynchronous Time Multiplexing of Speech Sources" - IRE Trans. Comun. Syst., Vol. CS-10, dezembro 1962, pp. 390-397.

[CHU 69] Chu, W.W. - "A Study of Asynchronous Time Division Multiplexing for a Time Sharing Computer System" - 2nd Annual Hawaii Conference, 1969.

[BRA 85] Brasileiro, M.A.G. e Moura, J.A.B. - "Interconexão Processador-Memória via Técnicas de Multiplexação por Divisão no Tempo em Sistemas e Multiprocessadores" - Grupo de Redes de Computadores da UFPb, RT-GRC/UFPb, abril 1985.

[BUZE 73] Buzen, J.P. - "Computational Algorithmus for Closed Queuing Networks with Exponential Servers" - ACM - setembro 1973 - pp. 527-531.

[DAVI 73] Davis, D.W. e Barber, D.L.A. - "Communication Networks for Computers" - John Willy & Sons, 1973.

[DOR 67] Dor, N.M. - "Guide to the Length of Buffer Storage Required for Randon Input and Constant Rates" -, IEEE Trans. on Eletronic Computers, outubro 1967, pp. 683-684.

[ENSL 77] Enslow, P.H. Jr. - "Multiprocessador Organization - A Survey", Computing Surveys, Vol. 9, Nº 1, março 1977 - pp. 103-129.

[FLYN 72] Flynn, M.J. - "Some Computer Organizations and their Effectiveness" - IEEE Trans. Comm. C-21, 9-1972 - pp. 948-960.

[JOEL 68] Joel, A.E. - "On Permutation Switching Networts" - BSTV maio-junho 1968, pp. 813-822.

[HAND 77] Handler, W. - "The Impact of Classification Schemes on Computer Architecture" - Proc. 1977 Intl. Conf. on Parallel Processing - agosto 1977 - pp. 7-15.

[LAWR 73] Lawrie, D.H. - "Memory-Processor Interconnection Networks" - Relatório da Universidade de Illinois UIUCDCS-R - fevereiro 1973, pp. 73-557.

[MEHR 79] Mehra, S.K. e Majithia, J.C. - "A Modified Ethernet for Multiprocessor Interconnection" - Proc. Comp. Networking Symposium, dezembro 1979 - IEEE - CH 1467 - pp. 132-136.

[PEAR 77] Pearce, R.C. - "Performance Studies of Parallel Computer Architecture" - Tese de Ph.D., University of Waterloo, Waterloo, Ontario, Canada - 1977.

[RUDI 71] Rudin, H.J. - "Performance of Simple Multiplexer-Concentrators for Data Communication" -, IEEE Trans. Comm., Vol. COM-19, Nº 2, abril 1971.

[STON 75] Stone, H.S. - "Introduction to Computer Architecture" - Science Research Associates, Inc. - 1975.

[TTL 76] TTL Databook 1976.

[WONG 75] Wong, J.W. - "Queueing Network Models for Computer Systems" - UCLA-ENG-7579 - School of Engineering and Applied Science, University of California at Los Angels - 1975.

B.B. SOTOMAYOR *
C.L. TOZZI **

SUMÁRIO

Este trabalho apresenta uma solução para o problema da entrada de dados gráficos em sistemas de PAC, utilizando técnicas de reconhecimento de padrões. São discutidos, de modo geral, os passos necessários para reconhecimento dos elementos de uma figura e descrita uma aplicação onde são identificadas portas lógicas e conexões entre as mesmas.

ABSTRACT

This paper presents a solution to the problem of inputting graphical data into CAD systems. The steps for recognition of the elements of a figure are discussed and applied to the identification of logical gates and their connection.

* Universidad de Tarapaca - Deptº de Eletrônica
Casilla 287 - Arica - Chile

**UNICAMP - Deptº Engenharia Elétrica - FEC
Cidade Universitária - 13.100 Campinas,SP

1. INTRODUÇÃO

Nos últimos anos vem sendo observado o emprego em escala cada vez maior do computador como ferramenta de auxílio a projetos nos mais diversos campos da engenharia, dando origem aos denominados sistemas de Projeto Auxiliado por Computador - PAC. Porém, em algumas aplicações, o ganho advindo da utilização do computador no auxílio ao projeto é prejudicado pela dificuldade de comunicação do usuário com o computador, podendo ser citadas as aplicações cujos dados se encontram na forma gráfica, por exemplo, desenhos feitos a mão pelo projetista, cuja descrição demandaria a definição de uma linguagem apropriada, e consequentemente, maior esforço para realização da entrada de dados.

Uma alternativa ao procedimento citado é dotar o sistema de auxílio a projetos com a capacidade de visão, de modo a se realizar a entrada dos dados diretamente do desenho. Dispor o computador da capacidade de visão, significa, na verdade, dispor de meios de digitalização da imagem (realizável através de câmeras de TV) e de programas, baseados nas técnicas de reconhecimento de padrões, que a partir da imagem digitalizada realize o mapeamento dos elementos da imagem num conjunto, previamente definido, de elementos permitidos.

Este trabalho, resultado de um programa de pesquisa desenvolvido no Departamento de Engenharia Elétrica da FEC - UNICAMP, discute de modo geral os passos necessários para o reconhecimento dos elementos de uma figura, descreve uma aplicação onde são identificadas portas lógicas e as conexões entre elas, e apresenta resultados e conclusões obtidas desta aplicação.

2. ETAPAS DE UM SISTEMA DE RECONHECIMENTO

O processo de reconhecimento tem início com a digitalização da imagem ou desenho, através de uma câmera de televisão, matriz de dados ou mesmo um tablet. Através da câmera de televisão obtém-se a discretização da imagem em diferentes níveis de cinza, enquanto que para os outros dois dispositivos a discretização é obtida com apenas dois valores. Ao processo de digitalização segue-se normalmente uma etapa de filtragem ou melhoramento da imagem, com aplicação, por exemplo, de técnicas de realçamento, redução de ruído, etc.

A etapa seguinte consiste na segmentação da imagem, isto é, a separação da figura em elementos isolados. Esta segmentação, de acordo com as ca-

racterísticas da figura, pode ser realizada através de métodos de detecção de bordas ou métodos de limiar ou thresholding. Ainda na etapa de segmentação pode ocorrer a aplicação de técnicas de enfraquecimento (thinning) com o objetivo de reduzir as dimensões da linha de contorno do objeto, numa preparação para a etapa de extração de atributo.

A fase de extração de atributos consiste na medida dos atributos dos elementos separados, os quais são usados na fase seguinte (fase de classificação) juntamente com regras de decisão, para se obter a associação do elemento à classe que o contém.

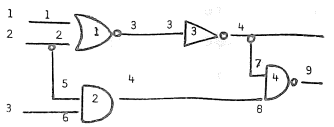
A seleção dos atributos e a definição das regras de decisão são realizadas previamente durante a fase de aprendizado. Nesta fase são estudados os atributos dos elementos de aplicação e selecionado o conjunto de atributos que permite uma discriminação real entre os diferentes objetos. O conjunto selecionado deve possuir baixa dimensionalidade e os atributos selecionados devem ser invariantes para posição, rotação e tamanho do objeto. Como atributos mais comumente usados podem ser citados: área, perímetro, circularidade, relação da distância do pixel ao contorno (fator G), coordenadas do centro de gravidade, número de Euler, ângulo de orientação, momentos, etc.

A definição das regras de decisão, que podem ser determinísticas ou estatísticas, é geralmente realizada de modo interativo. Uma amostra de objetos de uma classe é submetida ao extrator de atributos, sendo determinados o valor médio e a variancia de cada um dos atributos. O processo é repetido para todas as classes da aplicação, obtendo-se deste modo subsídios para a escolha das regras de decisão. Definidas as regras de decisão, uma amostra de teste é submetida ao sistema de reconhecimento e determinado o erro na classificação. Se o desempenho do classificador não é o desejado, novas regras deverão ser geradas ou mesmo novos atributos selecionados, de modo a obter-se o desempenho procurado.

Dentre os métodos mais usados para classificação através de regras determinísticas podem ser citados o classificador por distância mínima, que usa como critério de classificação a distância entre o vetor definido pelos valores dos atributos extraídos do padrão a ser reconhecido e o conjunto de vetores (um vetor associado a cada classe) obtidos da fase de aprendizado, e o classificador por função de discriminação linear, que tem como filosofia a definição de uma superfície de separação entre as diversas classes de aplicação.

3. DESCRIÇÃO DA APLICAÇÃO

A aplicação desenvolvida consiste no reconhecimento de portas lógicas e as conexões entre elas. Os resultados obtidos podem ser utilizados como entrada de dados para um simulador de circuitos digitais, ou como entrada de dados em um sistema de produção automática de documentação a partir de desenhos manuais. Na Figura 1 estão expressas as características dos dados de entrada para o sistema de reconhecimento e dos resultados obtidos.



Sistema de Reconhecimento de Padrões

IN 1.1
1.2
2.6
OUT 3.4
4.9
Ligações 1.3--3.3
2.4--4.8
3.4--4.7
1.2--2.5

Fig. 1 Aplicação do sistema de reconhecimento de padrões para entrada de dados nos exemplos citados.

As classes definidas para a aplicação consistem no conjunto de figuras de uso comum em projeto de circuitos digitais:

Classe 1 : AND
Classe 2 : OR
Classe 3 : N-INV
Classe 4 : NÓ
Classe 5 : NAND
Classe 6 : NOR
Classe 7 : INV

Para esta aplicação, o problema do reconhecimento é dividido em duas partes. Primeiro é feito o reconhecimento das portas lógicas e nós; a seguir, utilizando-se de dados obtidos na primeira fase, é feita a determinação das conexões de entrada e saída e das conexões entre portas lógicas.

3.1. AQUISIÇÃO E SEPARAÇÃO DAS CENAS

As cenas são obtidas através de uma mesa digitalizadora, com armazenamen

to em arquivos diferentes dos dados relativos aos elementos lógicos e conexões. Este procedimento adotado para aquisição de dados foi escolhido em função dos recursos disponíveis no laboratório; a aquisição de dados pode ser feita, também, através de uma câmera de TV, com utilização de características diferentes para os elementos lógicos e conexões (por ex. cor), sendo então os dados separados e armazenados em arquivos distintos, pela utilização de métodos de segmentação por limiar.

Os dados obtidos são tratados de modo a garantir o fechamento dos contornos dos elementos lógicos, a continuidade das conexões e demais características necessárias à correta extração de atributos.

3.2. DEFINIÇÃO DE ATRIBUTOS E REGRAS DE CLASSIFICAÇÃO (APRENDIZADO)

A partir de uma amostra padrão de cada uma das classes definidas para a

CLASSE 1	Variancia	Valor Médio	Val.Max.	Val.Min.
Circul.	.144770	1.52450	1.89700	1.46300
FatorG	87.1300	478.500	608.000	302.000
F.Mom1	1908E-6	1656E-4	1687E-4	1628E-4
F.Mom2	4221E-7	4497E-7	1438E-7	7120E-8
F.Mom3	2266E-7	3174E-7	8237E-7	1795E-7
F.Mom4	2065E-6	3007E-7	6834E-6	1962E-7
CLASSE 2	Variancia	Valor Médio	Val.Max.	Val.Min.
Circul.	.250760	2.47200	2.95410	1.97710
FatorG	213.970	507.500	812.000	225.000
F.Mom1	9855E-6	1932E-4	2143E-4	1836E-4
F.Mom2	9307E-7	1090E-6	3127E-6	9940E-8
F.Mom3	7730E-7	2691E-6	3904E-6	1694E-6
F.Mom4	7699E-5	1105E-5	2553E-4	2972E-6
CLASSE 3	Variancia	Valor Médio	Val.Max.	Val.Min.
Circul.	.405020	2.28080	3.14460	1.58220
FatorG	40.2020	105.500	186.000	64.0000
F.Mom1	9018E-6	1950E-4	2047E-4	1781E-4
F.Mom2	4794E-7	1067E-6	1496E-6	2449E-7
F.Mom3	1294E-6	4582E-6	6204E-6	2390E-7
F.Mom4	1582E-6	8604E-7	5130E-6	1488E-7

Tab.1 Valores de variancia, valor médio, valor máximo e mínimo para cada atributo da Classe 1 (AND), Classe 2 (OR), Classe 3 (INV).

aplicação são determinados os valores médios de circularidade, fator G, e funções de momentos, conforme apresentado na Tabela 1.

Verifica-se dos resultados da Tabela 1 que existe uma boa separação das classes a partir dos atributos tomados, sendo possível a definição das regras de decisão a partir destes atributos.

Para a classificação num primeiro nível utiliza-se o método da árvore binária, deste modo, a separação entre as classes 1, 2 , 3, 4 e 5, 6, 7, é obtida através da determinação da existência de cruzamentos nos contornos da figura.

Obtidos os dois primeiros grupos, a separação das classes de cada grupo é obtida através da utilização dos demais atributos, e usando como função de classificação a mínima distância. A Figura 2 mostra o esquema de classificação.

Deve-se ressaltar, que optou-se aqui pelo uso do classificador por mínima distância, porém, o processo completo de classificação poderia ser executado através do uso de uma árvore binária de decisões.

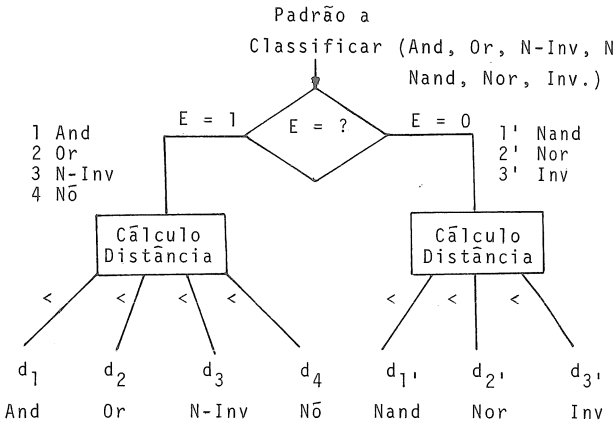


Fig. 2 Representação do esquema de classificação.

3.3. DETERMINAÇÃO DAS LIGAÇÕES ENTRE PORTAS

Para a determinação das ligações entre as portas são utilizados os seguintes atributos:

- coordenadas dos centro de gravidade da figura (CG)

- distância maior que existe do centro de gravidade ao contorno do objeto
- orientação da figura

Para cada elemento determinado na fase anterior, são medidos os dois primeiros atributos citados, e contruída uma tabela denominada Tabela de Elementos. Do arquivo de conexões são obtidas as coordenadas de início e fim de cada conexão, construída uma lista denominada Lista de Conexões e executado o seguinte algoritmo:

1. É realizada uma busca na Tabela de Elementos até a identificação de um nó:
   1.1. Com a coordenada do CG do nó é determinada na Tabela de Conexões a conexão correspondente ao nó.
   1.2. Na Tabela de Elementos e usando a Tabela 2 é verificada se a outra extremidade da conexão corresponde a uma ligação de saída ou entrada da porta.
   1.3. Se a ligação corresponde à saída da porta o nó é associado à porta e o par de coordenadas do nó é eliminado da Tabela de Conexões. Caso contrário, isto é, se a ligação corresponde à entrada da porta volte ao ponto 1.1.
   1.4. Retorne ao ponto 1, até que todos os nós tenham sido associados às portas.
2. Tomando um par de coordenadas da Tabela de Ligações são determinadas as portas associadas a esta ligação, considerando que a porta está contida num círculo definido pelo raio igual a maior distância que existe entre o centro de gravidade e o contorno da porta e pelo centro igual a coordenada do centro de gravidade da porta:
   2.1. Usando a Tabela 2 é determinado o tipo de ligação (entrada ou saída) para cada extremidade e definida a ligação entre portas. Se foi possível apenas a associação de uma das coordenadas à porta, então a outra extremidade corresponde à entrada ou saída externa do circuito, e sua característica é determinada pela outra extremidade.
   2.2. Volte ao ponto 2, enquanto ainda existem elementos a serem processados na Tabela de Ligações.

Orientação	$X_L > X_G$	$X_L < X_G$	$Y_L > Y_G$	$Y_L < Y_G$
0^o	Out	In		
90^o			In	Out
180^o	In	Out		
270^o		Out	In	

Tabela 2 Determinação para entrada e saída de portas.
X_G Y_G - coordenadas do centro de gravidade
X_L Y_L - Coordenadas extremas da conexão.

4. CONCLUSÕES

Os resultados obtidos para esta aplicação mostram que a entrada de dados em Sistemas de Auxílio a Projeto pode ser facilitada através do uso de técnicas de reconhecimento de padrões.

Para uma maior abrangência da aplicação discutida, devem ser incluídas novas classes para descrição de outros símbolos lógicos, tais como flip-flop, contadores, etc. bem como a possibilidade de reconhecimento de caracteres alfanuméricos para permitir a identificação de portas e ligações.

O trabalho desenvolvido pode ser usado em outras aplicações, dado que qualquer reconhecimento baseia-se nos mesmos princípios, variando apenas os atributos tomados para definição das regras de decisão. Uma área potencial para aplicação do estudo desenvolvido é a de visão de robôs.

5. BIBLIOGRAFIA

Barraza, S.B., "Reconhecimento de Padrões para Aplicações em CAD/CAM". Tese de Mestrado, UNICAMP, Dez.1984.

Gonzalez, R.C. & Safabakhsh, R., "Computer Vision Techniques for Industrial Applications and Robot Control". Computer, December 1982, 17-32.

Rosenfeld, A. & Kak, A.C., Digital Picture Processing. Academic Press, N.Y. , 1976.

Weszka, J.S., "A Survey of Threshold Selection Techniques". CGIP, 7,259-265 (1978).

Stefanelli, R. & Rosenfeld, R., "Some Parallel Thinning Algorithms for Digital Pictures". JACM, 18, 255-264 (1971).

Castleman, K., Digital Image Processing. Prentice-Hall, N.J., 1979.

Duda, R.O. & Hart, P.E., Pattern Clssification and Scene Analysis. John Wiley & Sons, Inc., N.Y., 1973.

Mascarenhas, N.D.A., "Processamento de Imagens, Classificação de Padrões e Visão Robótica". Tutoriais CONAI, 1983.

Kulpa, Z., "Area and Perimeter Measurement of Blobs in Discrete Binary Pictures". CGIP, 6, 434-451 (1977).

Wechsler, H., "An new and Fast Algorithm for Estimating the Perimeter of Objects for Industrial Vision Tasks". CGIP, 17, 375-385 (1985).

Wong, R.Y., "Scene Matching with Invariant Moments". CGIP, 8, 16-24 (1978).

Ballard, D.H. & Brown, C.M., Computer Vision. Prentice-Hall, 1982.

Meisel, W., Computer-Oriented Approaches ro Pattern Recognition. Academic Press, N.Y., 1972.

Tou, J.T. & Gonzalez, R.C., Pattern Recognition Principles. Addison-Wesley, Mass., 1974.

TIPOS DE DATOS ABSTRACTOS PARA LA ESPECIFICACION DE LA ANIMACION: SCRIPTS, ACTORS Y MOVIES

José Alberto Fernández*, Andreas Meier**

RESUMEN

En el presente trabajo se muestran algunos resultados de la aplicación de las técnica de especificación formal, mediante tipos de datos abstractos, al problema de la animación dentro de un sistema de producción de películas sintetizadas por computadora. En este sentido, se llega a la definición de tres tipos principales en el proceso de animación: Los Actores u objetos a ser animados, El Guión (Script) que define la animación y La Película (Movie) resultante.

ABSTRACT

In this paper we present some results concerning the application of the technique of formal specification using abstract data types to the problem of animation in the production of computer synthesized films. To this end, we define three principal types in the animation process: The Actors, or objects to be animated, The Script which defines the animation, and the resulting Movie.

*Profesor de la Universidad Simón Bolívar, Caracas, Venezuela.
Ingeniero de la Computación, USB. Areas de Interes: Computación Gráfica, Lenguajes de Programación e Inteligencia Artificial.

**Profesor de la Universidad Simón Bolívar, Caracas, Venezuela.
Phd en Ciencias de la Computación, Universidad de Ginebra, Suiza.
Areas de Interes: Computación Gráfica, Tratamiento de Imágenes e Inteligencia Artificial.

INTRODUCION

Uno de los aspectos menos investigados en el area de la Computación Gráfica, es el uso de las técnicas de especificación como base para el desarrollo de software. En realidad, el desarrollo de muchos sistemas se realiza en forma empírica desde el punto de vista de su concepción, con lo cual se llega a productos monolíticos en los cuales es dificil realizar cambios de diseño.

Por otro lado, la descripción imformal que se tiene de estos sistemas, trae como consecuencia, una mayor dificultad en el empleo de los mismos, pues no se posee una fuente de referencia para el implementador o usuario, clara y poco dependiente de una implementación particular.

Este tipo de problemas, ha sido enfrentado con exito en otras áreas de la computación (primordialmente en el área de Lenguajes de Programación) mediante el uso de las diversas metodologías de especificación formal que se han desarrollado.

En Computación Gráfica, se han comenzado a realizar esfuesos en esta vía y se ha llegado a algunos resultados importantes como los obtenidos por Mallgren [Mall82] que utiliza los Tipos de Datos Abstractos y los Tipos de Datos Compartidos (Shared Data Types) para especificar, no sólo los objetos que se manipulan sino tambien las primitivas de interacción que se utilizan en la comunicación usuario-sistema gráfico.

El objetivo de nuestra investigación, se centra en el uso de éste formalismo, en la concepción e implementación del módulo de descripción de la animación del Sistema de Producción de Peliculas Sintetizadas por Computadora que en estos momentos esta siendo desarrollado por el grupo de Computación Gráfica de la Universidad Simón Bolívar.

EL SISTEMA DE ESPECIFICACION FORMAL

El lenguaje formal utilizado, emplea las tecnicas de especificación algebraica de Guttag [Gutt75], Zilles [Zill74] y Goguen el al. [Gogu78] para describir los tipos de datos abstractos y sigue la sintáxis propuesta por Mallgren con ciertas modificaciones que facilitan la definición y uso de tipos de datos parametrizados.

Los operadores básicos de generación se reconocen en la especificación, por estar precedidos por un '·'. Las operaciones auxiliares, por estar precedidas en la firma por un '*'.

TIPOS BASICOS PARA LA DEFINICION DE OBJETOS JERARQUICOS

Debemos definir en primer lugar algunos tipos pictóricos básicos que son necesarios en nuestro sistema gráfico:

La estructura jerárquica a utilizar, es la siguiente:

Data Type Tree of Leaves and Nodes (tree(leaves,nodes), tr)

operators

·leaf	leaves X name	-> tree
·node	nodes X name	-> tree
·insprogeny	tree X tree	-> tree
insnode	tree X nodes	-> tree
delprogeny	tree X name	-> tree
getname	tree	-> name
getleaf	tree	-> leaves
getnode	tree	-> nodes
getprogeny	tree X name	-> tree
isleaf	tree	-> boolean

end

Un objeto gráfico jerárquico se define entonces como:

Data Type Picture-Tree (pictree, pt)

enrichment

pictree ≡ tree (spicture, grtransf)

extension

*display pictree -> picture

end

Esta operación de display define la visualización del objeto gráfico.

Los objetos elementales de los cuales están compuestos los pictrees, pueden ser curvas, superficies paramétricas o estructuras más simples que representen las componentes del objeto.

```
operations
    ·end                                          -> trajectory
   *·inscontrol     trajectory X item X knot      -> trajectory
     addcontrol     trajectory X item X knot      -> trajectory
     i_knot         trajectory X integer          -> knot
     getcontrol     trajectory X knot             -> item
    ·start          trajectory                    -> knot
     finish         trajectory                    -> knot
     interpolation  trajectory X knot             -> item
end
```

La operación de interpolation define el tipo de la interpolante, es por ello que sus axiomas serán definidos en cada enrequecimiento particular.

Si los objetos jerarquicos estan compuestos por curvas parametricas:

```
Data Type Simple Picture of Parametric Curve (spicture, sp)

enrichment
     spicture = trajectory (point, real)
extension
     *display  spicture              -> picture
end
```

Por ultimo:

```
Data Type Point (point, p)

operations
    ·point      real X real X real          -> point
     origin                                 -> point
     x_axis     point                       -> real
     y_axis     point                       -> real
     z_axis     point                       -> real
     sum        point X point               -> point
     diference  point X point               -> point
     distance   point X point               -> real
end
```

El proceso de descripción de la animación se ha dividido en varias partes:

- Definición de los grados de libertad y restricciones de los objetos a animar (creación de los actores)
- Definición de movimientos sobre sistemas articulados (creación del guión)

- Designación de actores para el guión (película)

Notese que las dos primeras etapas del proceso de animación son independientes, con lo cual es posible tener bibliotecas de actores y guiones, los cuales pueden asignarse con gran libertad.

Definición de los Actores

Los actores son objetos jerárquicos a los cuales se han asociado articulaciones en algunos de sus niveles de jerarquía.

Estas articulaciones pueden ser de dos tipos:

- Rotacionales:
  Los movimientos posibles consisten de rotaciones con respecto al sistema de coordenadas asociado a la articulación. La posición, nos indica los ángulos de rotación en los distintos ejes.

- Traslacionales:
  Los movimientos posibles son traslaciones de la articulación.

Definición del Guión

Los movimientos de una articulación se definen mediante interpolación. Se basan en los trabajos de Bartels y Kochanek [KoBa84] referentes al uso de parámetros de continuidad, bias y tensión sobre interpolantes Hermite con el fin de obtener mayor expresividad temporal en la curva interpolada.

Un script, será entonces una estructura jerárquica de movimientos, los cuales indican el valos de las articulaciones en cada instante de tiempo.

Asignación del Script al Actor

La asignación de un Script a un Actor, debe tomar en cuenta tanto las restricciones que el actor posee, como la estructura jerárquica del mismo. Esto es:

- La jerarquía del Actor debe ser compatible con la del Script.

- Ningún movimiento definido en el Script debe violar las restricciones que posee la articulación.

Debemos definir el significado de compatibilidad utilizado aqui:

Sea A el conjunto de articulaciones presentes en el Actor, y $(A,<=)$ el conjunto parcialmente ordenado definido por la jerarquía de las articulaciones.

Sea M el conjunto de movimientos presentes en el Script, y $(M,<<=)$ el conjunto parcialmente ordenado definido por la jerarquía de los movimientos.

Sea D una función de M en A que asigna el Script al Actor.

$$F: M \rightarrow A$$

Llamemos M_D al dominio de F, M_D subconjunto de M.
Llamemos A_D a la imagen de M_D bajo F, A_D subcunjunto de A.

Definimos que la designación del Script es compatible con el Actor, si:

$(A_D,<=)$ es un CPO, $(M_D,<<=)$ es un CPO y F define un isomorfismo de $(M_D,<<=)$ en $(A_D,<=)$.

Especificación del Actor

Veamos algunas de las firmas de los tipos componentes de Actor:

Data Type Articulation of Actors (articulation, ar)

operations

·rigid		-> articulation
·traslartic		-> articulation
·rotatartic		-> articulation
·insrestriction	articulation X restriction X name	-> articulation
getrestriction	articulation X name	-> restriction
traslation	articulation	-> point
rotation	articulation	-> rotator
istraslrestricted	articulation X point	-> boolean
isrotatrestricted	articulation X rotator	-> boolean
setrotat	articulation X rotator	-> articulation
settrasl	articulation X point	-> articulation

end

Las restricciones serán expresadas como curvas que definen la frontera de un volumen donde es puede mover la articulación.

Data Type Restriction of Articulation (restriction, rt)

enrichment

restriction ≡ trajectory (point, real)

extentions

frontier restriction -> region

end

La operación frontier nos permite conocer la región que define el la superficie de la interpolante, y de esta forma podremos verificar si la restricción se respeta o no, al intersectarla con la trayectoria del movimiento a verificar.

Data Type Rotator (rotator, rt)

operations

·rot	real X real X real	-> rotator
origin		-> rotator
xy_angle	rotator	-> real
z_xyangle	rotator	-> real
spin	rotator	-> real
sum	rotator X rotator	-> rotator
diference	rotator X rotator	-> rotator
transfor	rotator	-> geomfcn
pointof	rotator	-> point

end

La operacion **transfor** es la encargada de indicar cual es la transformación geométrica asociada a la posición actual de la articulación. **pointof** efectua una transformación del espacio esferico en que se expresan los **rotators** al espacio cartesiano de los puntos.

Una vez especificados estos tipos, podemos definir a un actor como un Tipo Abstracto que posee la siguientes características.

Data Type Tree Structured Actor (actor, ac)

enrichment

actor ≡ tree (pictree, pair (articulation, grtransf))

extensions

actor	pictree	-> actor
addart	actor X articulation X name	-> actor
getartic	actor	-> articulation
gettransf	actor	-> grtransf
setartic	actor X articulation	-> actor
getpos	actor	-> grtransf
isrigid	actor	-> boolean
*display	actor	-> picture

end

En el proceso de construcción del actor pueden existir nodos de la figura que no poseen ninguna articulación, aunque son ancestros de nodos que contienen articulaciones. En estos casos dichos nodos tendrán una

articulación rígida. Notese que bajo el concepto de actor, se puede incluir a la escena que se esta animando, formando cada uno de los diversos objetos que en ella se mueven, una parte de un objeto mayor.

Especificacion del Script

La base del Script son los movimientos, veamos como estan definidos:

```
Data Type Movement of Articulations (movement, mm)

operations
     *rotmov     animation (rotator) -> movement
     *traslmov   animation (point)   -> movement
      istrasl    movement            -> boolean
      start      movement            -> time
      finish     movement            -> time
end

Data Type Animation Script (script, sr)

enrichment
      script = tree (nulltype, pair(time,movement))
extension
      attime     script X time       -> script
      insmov     script X movement   -> script
      entry      script              -> time
      move       script              -> movement
end
```

Como se puede notar los movimientos colocados en el Script se efectuarán en el instante de tiempo especificado, relativo al inicio de dicho movimiento en el script; la operación attime y entry permiten colocar e inspeccionar, respectivamente, el tiempo de inicio del movimiento.

Especificación de la película

```
Data Type Movie (movie, mv)

enrichment
      movie = tree (actor, triple (time, movement, grtransf))
```

```
extension
     movie      script X actor       -> movie
     addmovie   script X movie       -> movie
     iscompat   script X actor       -> boolean
     start      movie                -> time
     finish     movie                -> time
     ontime     movie X time         -> actor
    *display    movie X time         -> picture
end
```

El proceso de animación se puede ver como la evaluación de la operación de display cada $^1/_{30}$ de segundo.

ASPECTOS DE LA ESPECIFICACION AUN POR RESOLVER

Hasta el momento, hemos logrado especificar los tipos que componen el Módulo de Animación, faltan por tratar los aspectos referentes a la interface con el usuario, en particular:

Especificación del proceso de construción de los actores, movimientos, y script.

Especificación del lenguaje de control global de la película; es necesario poder realizar cambios de alto nivel sin tener que modificar los objetos que componen la película. (Lenguaje de Coordinación. Ej.: DIAL [FeSB82])

CONCLUSIONES

Se ha podido comprobar que la metodología de especificación propuesta por Mallgren permite especificar gran parte de los aspectos referentes al proceso de definición de imagenes tridimensionales (en su trabajo sólo analiza el problema 2-D).

El uso de tipos abstractos en la especificación del sistema ha permitido una definición clara y modular. Un cambio importante, por ejemplo, el tipo de objetos jerárquicos que se tienen o la forma en que las restricciones son especificadas, conlleva solamente el cambio de ciertos operadores dentro de la especificación que hemos realizado. De la misma

forma, un cambio en la manera en que los movimientos son especificados, se traduce en la modificación de pocas operaciones.

BIBLIOGRAFIA

[FeSB82] Feiner, S., Salesin, D., and Banchoff, T. "DIAL: A Diagramatic Animation Language", IEEE Computer Graphics & Animation, September 1982.

[Gogu78] Goguen, J.A., Thatcher, J.W., and Wagner, E.G. "An initial algebra approach to the specification, correctness and implementation of abstract data types, in Yeh", R.T. (ed) Current Trends in Programming Methodology, vol. IV, Data Structuring, Prentice Hall, Englewood Cliffs, New Jersey, 1978.

[Gutt75] Guttag, J.V. "The specification and application to programming of abstract data types", Tech. Report CSRG-59, University of Toronto, Toronto, Canada, September 1975.

[KBad82] Korein, J.U., and Badler, N.I. "Techniques for Generating the Goal-Directed Motion of Articulated Structures", IEEE Computer Graphics & Animation, September 1982.

[KoBa84] Kochanek, D., Bartels, R. "Interpolating Splines with Local Tension, Continuity, and Bias Control", Procedings Siggraph'84 July 84, 18(3), 33-41.

[Mall82] Mallgren, W. "Formal Specification of Interactive Graphics Programming Languages". ACM Distinguished Doctoral Dissertation Series, The MIT Press 1982.

[Zill74] Zilles, S.N. "Algebraic specification of data types, Project MAC Progress Report 11 (1974), Massachusetts Institute of Technology, Cambridge, Mass.,
52-58.

O MAPEAMENTO DE TEXTURA NA SÍNTESE DE IMAGENS REALISTAS

C.C. Alvarenga* R.N.Melo**

Resumo:

Este trabalho aborda as técnicas de Mapeamento de Textura como uma alternativa na síntese de imagens realistas. Inicialmente apresenta um histórico das várias técnicas propostas na literatura, depois estabelece critérios de comparação e finalmente apresenta um quadro comparativo sugerindo uma classificação.

Abstract:

This paper approaches Texture Mapping Techniques as an alternative in realistic image synthesis. At first it presents historically the many techniques available in literature, later it defines comparison criteria between these techniques. Last, it presentes a comparison table which suggests a classification of these techniques.

* M.Sc 1984 Depto. Inf. PUC/RJ, Analista de Sistemas TV Globo, áreas de interesse: Computação Gráfica, Banco de Dados

** D.Sc 1976 ITA, Prof. Associado Depto. Inf. PUC/RJ, área de interesse: Computação Gráfica, Banco de Dados e Engenharia de Software.

INTRODUÇÃO

Um dos objetivos da aplicação de técnicas de Computação Gráfica Raster em várias áreas é a geração de imagens sintéticas de objetos 3D que se confundam com imagens de objetos reais. As técnicas de Mapeamento de Textura simulam a textura das superfícies reais. A partir do trabalho de Edwin Catmull, que em 1974 introduziu esta idéia, várias técnicas de Mapeamento de Textura tem sido propostas.

O presente trabalho faz um estudo comparativo das técnicas de Mapeamento de Textura disponíveis na literatura visando conhecer as semelhanças, diferenças ou relações entre elas. Para isso são identificados os critérios para a comparação das técnicas e com base neles elas são comparadas e classificadas.

O MAPEAMENTO DE TEXTURA

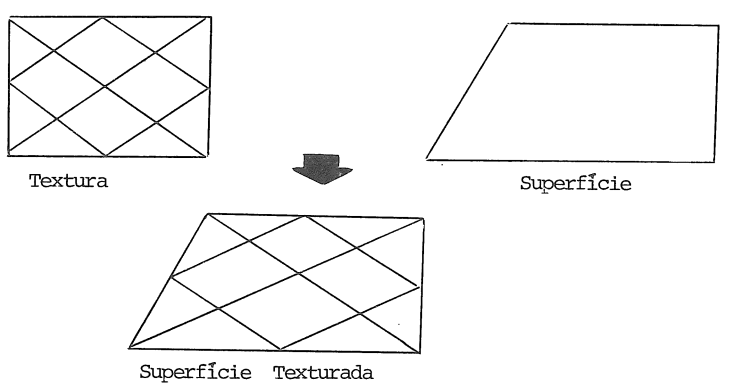
A pesquisa na área de Computação Gráfica orientada para a geração de imagens realistas está em andamento desde meados da década de 60, e tem produzido imagens bastante impressionantes. O objetivo dessa pesquisa tem sido produzir imagens tão realistas que um observador confunda a imagem de um objeto sintético definido apenas pelo seu modelo matemático, com a imagem de um objeto real. Esse é um objetivo importante em áreas como Simulação, Projeto e Entretenimento.

Dada uma coleção de superfícies 3D, que representam as faces de objetos de uma cena com posições e orientações definidas no espaço, e dada a localização do observador e a direção de visualização desejada, pode-se gerar uma imagem da cena. A geração de imagens realistas requer também a consideração das propriedades das superfícies e da iluminação que as atinge. As propriedades da superfície podem incluir a sua cor, transparência, brilho, entre outras. Nos sistemas de geração de imagens mais simples essas propriedades são constantes para um mesmo elemento de superfície. A intensidade luminosa de um ponto numa superfície é determinada geralmente pela soma das componentes de reflexão da luz ambiente e da luz de fontes pontuais considerando os coeficientes de reflexão por difusão e especular. Técnicas mais elaboradas, entre elas as de Mapeamento de Textura, permitem que essas

propriedades variem nesses elementos básicos, aprimorando o realismo da imagem obtida.

Uma crítica frequente às tentativas iniciais de síntese de imagens matizadas realistas era relacionada à aparência das superfícies geradas. Inicialmente as superfícies pareciam foscas e nubladas, depois elas adquiriram brilho e transparência com técnicas mais elaboradas. Entretanto, as superfícies reais tem variações locais altamente complexas conhecidas como textura. Essas variações são de difícil modelamento pelos meios convencionais pois requerem muitos vértices ou pontos de controle para reproduzir precisamente a superfície. O Mapeamento de Textura aumenta consideravelmente o realismo e o conteúdo de informação dessas imagens e possibilita a produção de imagens de grande complexidade sem o consequente aumento de complexidade correspondente na descrição geométrica dos objetos.

As técnicas de Mapeamento de Textura simulam as irregularidades das superfícies reais em vez de descrevê-las com um modelo geométrico. A textura é armazenada como uma imagem digital e cada ponto é acessado por um par de índices. As intensidades em torno de cada ponto da textura são mapeadas no ponto correspondente da superfície como é ilustrado abaixo:



Mapeamento de Textura

A descrição de uma cena é contínua, e geralmente fornecida em termos de certas primitivas geométricas, como linhas, polígonos, - "patches" (elementos curvos de superfície), etc. A imagem produzida nos dispositivos raster consiste, então, de uma matriz de amos - tras da função contínua desejada. Essa amostragem acarreta numa grade variedade de efeitos visuais indesejados.

Um intervalo de amostragem inadequado na síntese de imagens digitais gera problemas de "aliasing". Infelizmente, os limites tecnológicos e a padronização dos equipamentos nos impedem de escolher o intervalo de amostragem. Mesmo que isso fosse possível, alguns problemas ainda permaneceriam. O pós-processamento da imagem também não resolve todos os problemas (CROW81, CROW83).

A solução para esse problema é filtrar a imagem contínua espacialmente para eliminar informação de alta frequência, antes de amostrar a imagem. Visualmente o efeito é nublar a imagem; matematicamente isto significa que a intensidade de um pixel não deve ser apenas a intensidade do ponto correspondente no espaço imagem e sim uma média ponderada das intensidades que circundam este ponto.

Quando uma textura é mapeada numa superfície ela deve ser esticada e cumprimida em alguns locais de modo a se ajustar à forma da superfície. A perspectiva também distorce as texturas mapeadas numa superfície. Assim, a menos que a textura seja bastante suave, os efeitos de "aliasing" se tornam um problema importante a resol - ver.

HISTÓRICO DAS TÉCNICAS DE TEXTURAMENTO

Em 1974, Edwin Catmull introduziu a idéia de simular as irregularidades das superfícies reais (CATM74). Ele concebeu e imple - mentou o primeiro sistema a usar imagens de texturas aplicadas a superfícies para dar o efeito de uma textura real (CROW84).

Dando continuidade ao trabalho de Catmull, Blinn e Newman propuseram um método que faz um mapeamento inverso do pixel a ser texturado na textura propriamente dita (BLIN76). Ele utiliza um filtro de "antialiasing" melhor que a amostragem por área, usada por / Catmull, e que não requer memória adicional.

Analogamente à técnica de Blinn, Feibush et al transforma do espaço imagem no espaço de definição da textura cada pixel de um polígono a ser exibido compensando a distorção da perspectiva / (FEIB80). Esta técnica facilita a comparação entre vários filtros, porque as funções de ponderação que definem a forma dos filtros são pré-computadas e armazenadas em tabelas de consulta.

Num enfoque um pouco diferente, Dungan et al obtém a aparência texturada de um objeto modelado por polígonos através da repetição sobre a superfície, de azulejos de textura em perspectiva pela superfície (DUNG78). Para diminuir o tempo de computação devido aos problemas de "aliasing". Neste método, a textura é pré-filtrada gerando imagens com diferentes resoluções, ou níveis de detalhe.

Considerando os problemas de "aliasing" na animação, ou seja, considerando que as dificuldades de amostragem inerentes ao processo de Mapeamento de Textura podem se agravar à medida que a vista da superfície muda, William propôs a criação de um conjunto de imagens fonte pré-filtradas representadas numa estrutura de dados piramidal paramétrica (WILL83). Essa estrutura minimiza os efeitos de "aliasing" e assegura a continuidade dentro e entre imagens. Porém, a filtragem é simétrica porque cada nível pré-filtrado é filtrado igualmente em X e Y.

Crow generalizou a técnica de Williams fornecendo uma aproximação melhor para a intensidade da textura permitindo que sejam utilizadas regiões retangulares da textura e não apenas regiões quadradas, usadas por Williams, o que implica numa compressão igual em X e Y, ou seja, uma filtragem simétrica (CROW84). Na técnica de Crow é utilizada uma tabela de áreas somadas, a partir da qual uma extensão virtualmente contínua de densidade de textura pode ser selecionada.

O objetivo do presente trabalho é fazer um estudo comparativo das técnicas de Mapeamento de Textura disponível na literatura procurando identificar os seus tópicos principais visando o melhor entendimento do problema e a orientação na seleção da técnica adequada a um dado ambiente. Na próxima seção, os critérios para essa comparação são identifi-ados e em seguida um quadro comparativo é apresentado. No final, o trabalho sugere uma classificação das técnicas.

As técnicas de Mapeamento de Textura são utilizadas na geração de imagens realistas ou artísticas em diversas áreas. A produção de imagens complexas por computador requer geralmente a interação de vários programas diferentes que modelam os objetos e implementam a sintetização de suas imagens. Portanto, para estabelecer os critérios de comparação, além de considerar o que são essas técnicas deve-se também considerar outros tópicos de natureza computacional e de realismo de cena complementares no processo todo.

A técnica de Mapeamento de Textura é uma técnica de matização que procura ajustar uma imagem digital (a textura) na superfície de um objeto 3D que é projetado num plano para visualização. Quando uma textura é mapeada numa superfície ela deve ser esticada e comprimida em alguns locais de modo a se ajustar à forma da superfície. / Assim, os efeitos de "aliasing" logo aparecem como um problema importante a resolver. Por isso todas as técnicas estudadas apresentam um esquema de filtragem para a solução desses problemas, associado ou não à uma compensação da distorção produzida pela perspectiva. Além disso, a maioria das técnicas requer memória adicional de forma a tornar a filtragem mais eficiente.

Os tópicos acima considerados podem servir de critérios para a comparação das técnicas analisadas. Mais especificamente esses critérios são:

Filtragem - É a principal característica que distinguem as técnicas, sendo analisada quanto ao tipo de filtro empregado, quanto ao instante de sua aplicação e quanto ao sobrecusto de memória. O tipo identifica o filtro empregado. O instante de aplicação especifica se o processo de filtragem consiste de uma filtragem prévia da textura e um esquema de indexação das estruturas intermediárias geradas ou se esse processo envolve a filtragem imediata dos elementos da textura que contribuem para a intensidade de cada pixel do elemento de superfície considerado. O sobrecusto de memória se refere à necessidade ou não do gasto de memória adicional para a implementação da filtragem.

Compensação da perspectiva - Este tópico identifica se foi utilizado ou não algum esquema para compensar a distorção introduzida pela projeção perspectiva necessária para a visualização realista de objetos 3D.

Por serem específicos do processo de texturamento esses tópicos acima foram classificados no quadro comparativo como tópicos específicos. Os tópicos a seguir foram classificados como complementares pois embora sejam imprescindíveis eles complementam os tópicos acima no processo de texturamento.

Modelo geométrico - Especifica o modelo geométrico dos elementos de superfície dos objetos a serem texturados.

Algorítmo para remoção das superfícies escondidas - Especifica o algorítmo utilizado no processo de síntese para a remoção das superfícies escondidas.

QUADRO COMPARATIVO

O quadro comparativo mostrado na figura 1 foi construído baseado nos critérios estabelecidos na seção anterior. Nesse quadro, as técnicas foram também organizadas cronologicamente.

TÓPICOS / TÉCNICAS	ESPECÍFICOS			COMPLEMENTARES		
	FILTRAGEM			COMPENSAÇÃO PERSPECTIVA	MODELO GEOMÉTRICO	ALGORÍTIMO PARA REMOÇÃO DAS SUPERFÍCIES
	TIPO	INSTANTE DE APLICAÇÃO	SOBRECUSTO DE MEMÓRIA			
CATM74	Fourier	Imediata	Pouco	Não	"Patch"	Buffer-z
BLIN76	Pirâmide	Imediata	Não	Sim	"Patch"	Linha de Exploração
DUNG78 (1)	Fourier	Prévia	Muito	Sim	Polígonos	"?" (2)
	Pirâmide	Imediata				
FEIB80	Qualquer	Imediata	Pouco	Sim	Polígonos	Subdivisão de polígonos
WILL83	Qualquer	Prévia	Muito	Sim	"?" (2)	Linha de exploração
CROW84	Fourier	Prévia	Muito	Sim	"?" (2)	Linha de exploração

Figura 1 - Quadro comparativo das técnicas de mapeamento de textura

(1) - A técnica de Dungan et al subdivide cada pixel em "subpixels" e os processa individualmente aplicando uma filtragem de janela de Fourier prévia da textura para definir os seus valores de intensidade. Além disso, as intensidades desses "subpixels" são integradas através de uma filtragem por uma função pirâmide para a determinação do valor final. Por isso esta técnica contém duas especificações de filtragem.

(2) - Os "?" indicam que não foi possível concluir de forma precisa a partir das referências bibliográficas.

ANÁLISE DAS TÉCNICAS

As técnicas de Mapeamento de Textura se aplicam tanto à geração de sequências animadas como também à produção de imagens isoladas, realistas ou artísticas. Como a qualidade da filtragem é proporcional ao tempo de processamento, a qualidade das imagens de sequências animadas pode ser comprometida já que são necessárias, em média, 30 imagens por segundo de animação. Assim, embora todas

as técnicas procurem gerar boas imagens de forma eficiente, algumas dão mais ênfase à qualidade da filtragem enquanto que outras à eficiência do processo de texturamento, visando a animação dessas imagens. Portanto, inicialmente, serão abordadas cada uma dessas duas classes de técnicas isoladamente. Em seguida, através de uma análise de cada tópico, essas técnicas serão comparadas em conjunto. Finalmente, serão comentados aspectos gerais relevantes que não foram abordados nas análises anteriores.

Qualidade da filtragem x Eficiência do processo

As técnicas de Catmull (CATM74), Blinn e Newell (BLIN76) e Feibush et al (FEIB80) são indicadas para a geração de imagens de alta qualidade já que podem utilizar filtros mais sofisticados aplicados imediatamente. A técnica de Catmull, pioneira, não considera nenhuma compensação para a distorção perspectiva, gerando imagens de menor qualidade que as outras. A grande vantagem da técnica de Feibush et al é a flexibilidade na utilização dos filtros já que a memória adicional requerida é uma tabela de consulta que armazena coeficientes que definem a forma do filtro desejado. Entretanto, esta e a técnica de Blinn (BLIN76) são bastante caras em tempo de processamento (CROW84).

Dungan et al (DUNG78), Williams (WILL83) e Crow (CROW84) - criaram técnicas orientadas para a geração de sequências animadas uma vez que utilizam filtros simples aplicados previamente. Visando melhorar a qualidade da imagem, Dungan et al aplicam também uma filtragem imediata além da prévia. Williams utiliza um filtro janela de Fourier simples na filtragem de cada nível da textura, mas qualquer filtro mais sofisticado pode ser utilizado. Essa flexibilidade é perdida na técnica de Crow, que não requer que a compressão seja simétrica como Williams, já que pode recuperar áreas retangulares, associadas a cada pixel dos elementos de superfície, da sua tabela de áreas somadas. Entretanto, geralmente, a técnica de Crow é mais eficiente (CROW84).

Análise dos tópicos

a) Filtragem - Como já foi observado, a aplicação da filtragem identifica se o objetivo da técnica é qualidade ou eficiência.

Uma das técnicas (DUNG78), visa a eficiência, mas também aplica uma filtragem imediata. Uma prática comum, observada das referências, que independe do objetivo, é a filtragem prévia das texturas que contenham muitas componentes de alta frequência. A filtragem através de uma janela de Fourier, embora simples, é bastante utilizada.

Apenas a técnica de Blinn (BLIN76) não requer memória adicional. A palavra de memória adicional por pixel requerida por Catmull (CATM74) especificando a área que os fragmentos de "patch" cobrem, representa, sem dúvida, um sobrecusto razoável de memória. A tabela de consulta de Feibush et al (FEIB80) é um sobrecusto de memória pequeno, pois contém apenas os coeficientes que definem a forma do filtro. As estruturas de Dungan et al (DUNG78), Williams (WILL83) e Crow (CROW84) contém intensidades que armazenam a imagem em diferentes resoluções e representam um sobrecusto de memória bastante grande.

b) Compensação perspectiva - Apenas a primeira técnica publicada (CATM74) não considerou a distorção da perspectiva. Podemos concluir que, atualmente, todo algorítmo de Mapeamento de Textura deve incluir o tratamento desse tópico. Além disso, observa-se que em função da eficiência do processo as técnicas que visam a geração de sequências animadas simplificam a definição dos elementos da textura que contribuem para a intensidade de cada pixel considerado comprometendo a qualidade da imagem.

c) Modelo geométrico - Em geral as técnicas publicadas para o mapeamento de texturas em superfícies usam modelos de polígonos ou "patches".

d) Algorítmo para remoção das superfícies escondidas - Geralmente, a matização é processada junto com a remoção das superfícies escondidas. Uma vantagem decorrente da separação desses processos, proposta por Feibush et al (FEIB80) é o processamento sequencial dos arquivos de textura, importante quando existem várias texturas diferentes numa mesma imagem.

Comentários finais

Uma análise cronológica das técnicas indica que uma preocupação atual é a pesquisa de técnicas que dem mais eficiência ao processo. Em 1980, num enfoque bastante diferente, Catmull e / Smith (CATM80) apresentaram uma técnica que transforma a imagem digital como uma imagem 2D que se ajusta à projeção de um polígo no colocado arbitrariamente no espaço 3D. A grande vantagem apresentada é que a transformação 2D pode ser decomposta em duas transformações simples, uma na ordem de exploração de linha vertical e a outra na horizontal. O processamento de pixels na ordem de exploração de linha tanto na geração da imagem transformada quanto na exploração da imagem original, viabiliza a implantação dessa técnica também em hardware.

CONCLUSÕES

A partir do trabalho de Edwin Catmull, que em 1974 introduziu esta idéia de simular as texturas das superfícies reais, várias técnicas de Mapeamento de Textura tem sido propostas. Estudos das técnicas de Mapeamento de Textura disponíveis na literatura permitem a identificação de tópicos diferenciadores que dão uma orientação na seleção da técnica adequada a um dado ambiente.

Identificados os tópicos que servem de critérios para a comparação das técnicas e, considerando a integração do processo de texturamento com a síntese de imagens realistas, pode-se classificar esses critérios como: complementares ao texturamento e específicos do processo. Os critérios considerados específicos foram : filtragem e compensação perspectiva. Os complementares são: modelo geométrico e algorítmo para remoção das superfícies escondidas.

Considerando que as técnicas de Mapeamento de Textura procuram ajustar uma imagem digital na superfície de um objeto 3D cuja projeção num plano de visualização é amostrada para a geração da imagem final. Este trabalho também identificou o tratamento da filtragem como a principal característica que distingue as técnicas. A análise destas técnicas à luz dos critérios específicos,- de acordo com o quadro comparativo construído, levou à classificação das técnicas em duas classes, em função do objetivo dessas técnicas. Elas enfatizam ou a qualidade da imagem ou o tempo do

processo de texturamento, visando a geração de sequências animadas de imagens. As primeiras utilizam filtros mais sofisticados aplicados imediatamente, e procuram soluções mais exatas à compensação da distorção perspectiva. As técnicas mais preocupadas com a eficiência, mais atuais devido provavelmente à maior propagação de aplicações que requerem animação, se utilizam de filtros mais simples aplicados previamente e em função do tempo de processamento comprometem a precisão na compensação da perspectiva. Ambas as classes se utilizam de memória adicional para agilizar o processo.

Esse trabalho está integrado a linha de pesquisa de Computação Gráfica e PAC (Projeto Assistido por Computação) do Departamento de Informática da PUC/RJ, especialmente na aplicação de Tecnologia Raster na geração de imagens realistas em diversas áreas tais como Engenharia, Arquitetura e Artes.

REFERÊNCIAS BIBLIOGRÁFICAS

BLIN76 Blinn, J. F. e Newell, M. E., "Texture and Reflection on Computer Generated Images", Communications of the ACM, 19(10), out 1976, pp 542-547.

BLIN78 Blinn, J. F., "Computer Display of Curved Surfaces", Ph.D. Thesis, University of Utah, Dept. of Computer Science, dez 1978.

CATM74 Catmull, E., "A Subdivision Algorithm for Computer Display of Curved Surfaces", Ph.D. Thesis, University of Utah, Dept. of Computer Science, dez 1974.

CATM80 Catmull, E. e Smith, A. R., "3-D Transformation of Images in Scanline Order", Computer Graphics (Proc. Siggraph '80), jul 1980, pp 279-285.

CROW76 Crow, F. C., "The Aliasing Problem in Computer Synthesized Shaded Images", Ph.D. Thesis University of Utah, Dept. of Computer Science, mar 1976.

CROW81 Crow, F. C., "A Comparison of Antialiasing Techniques", IEEE Computer Graphics and Applications, 1(1), jan 1981, pp 40-48.

CROW83 Crow, F. C., "Computational Issues in Rendering Antialiased Detail", Notas do Seminário "Advanced Image Synthesis" de Siggraph '83, jul 1983.

CROW84 Crow, F. C., "Summed-Aerea Tables for Texture Mapping", Computer Graphics (Proc. Siggraph '84), 18(3), jul 1984, pp 207-212.

DUNG78 Dungan Jr, W., Stenger, A. e Sutty, G., "Texture Tile Considerations for Raster Graphics", Computer Graphics, 12(3), ago 1978, pp 130-134.

FEIB80 Feibush, E. A., Levoy, M., e Cook, R. L., "Synthetic Texturing Using Digital Filters", Computer Graphics (Proc. Siggraph '80), 14(3), jul 1980, pp 294-301.

FOLE82 Foley, J. D., e Van Dam, A., "Fundamentals of Interative Computer Graphics", Addison-Wesley, Reading, MA, 1982.

GREE83 Greenberg, D. P., "Tutorial: Introduction to Raster Graphics", notas do Seminário "Introduction to Raster Graphics" de Siggraph '83, jul 83.

NEWM79 Newman, W. M. e Sproull, R. F., "Principles of Interactive Computer Graphics", McGraw-Hill, New York, 1979.

SUTH74 Sutherland, I. E., Sproull, R. F. e Schumacker, R. A., "A Characterization of Ten Hidden Surface Algorithms", ACM Computing Surveys, 6(1), mai 1974, pp 1-55.

WILL83 Williams, L. "Pyramidal Parametrics", Computer Graphics (Proc. Siggraph '83), 17(3), jul 1983, pp 1-11.

WHIT82 Whitted, T., "Processing Requirements for Hidden Surface Elimination and Realistic Shading", Notas do Seminário "Advanced Image Synthesis" de Siggraph '82, jul 1982.

INFORMATICA

C. GUICHOU*

J. BARRERA**

SUMÁRIO

A extração automática de parâmetros geométricos nas imagens digitais não é geralmente realizável num único passo. A abordagem da Morfologia Matemática consiste numa modificação progressiva da imagem antes da fase de medição. São apresentadas transformações morfológicas básicas (baseadas no conceito da erosão) e um exemplo de obtenção de um atributo de forma, o perímetro.

ABSTRACT

Automatic extration of geometrical parameters in digital images is not usually achieved in a unic step. The Mathematical Morphology approach consists in a progressive modification of the image before the measuring stage. Morphological basic transformation (based upon the concept of erosion) and an example of shape feature determination, the perimeter, are presented.

* Doutor de Terceiro Ciclo em "Astronomia e Técnicas Espaciais" (Universidade Paul Sabatier, Toulouse, França); Cooperante francês no INPE-Instituto de Pesquisas Espaciais, Caixa Postal 515, São José dos Campos-SP, CEP 12200.

** Engenheiro Eletricista (Politécnica, USP, 1983); Assistente de Pesquisa no INPE - Instituto de Pesquisas Espaciais, Caixa Postal 515, São José dos Campos-SP, CEP 12200.

INTRODUÇÃO

A análise de imagens ocupa uma parte importante na área de visão por computador. Distinta da fase de pré-processamento, que geralmente envolve operações lineares (filtragem, convolução), e das técnicas de classificação que usam regras de decisão estatísticas, a análise de imagens (Serra, 1982) se apoia na geometria integral (Kendall and Moran,1963; Santaló, 1976 e Matheron, 1975) e na álgebra booleana para alcançar os seguintes objetivos:

1) extração dos parâmetros quantitativos das zonas interessantes da imagem (sejam topológicos, tais como o número de conectividade, ou medidas individuais relativas a cada partícula, isto é, superfície, perímetro, etc.);

2) extração de zonas da imagem que possuem parâmetros característicos, principalmente atributos de forma.

As aplicações dessas técnicas já são operacionais em várias áreas: imagens microscópicas de medicina (Preuteux, 1984; Beucher, 1984), de metalografia (Serra, 1968) e recentemente de visão robótica, de acordo com Sternberg, 1982 (apud Serra, 1983).

Na Morfologia Matemática (Serra, 1982), os objetos que compõem a imagem E são percebidos como subconjuntos do conjunto $P(E)$ das partes de E. Geralmente, em processamento de imagens, E é incluído em $\mathbb{R}^2$, mas pode ser em $\mathbb{R}^3$ (volumes), e $\mathbb{R}^n$, ou, utilizando níveis de cinza, em $\mathbb{R}^{n+1}$, onde a tonalidade é então considerada como uma dimensão suplementar. As leis que governam a percepção visual, como por exemplo: "o objeto X_1 ficando atrás do objeto maior X_2 é invisível" são as leis de interseção ($\cap$) e da união ($\cup$). Esta observação simples, devida à opacidade da maioria dos objetos, justifica um princípio fundamental na análise de imagens: as operações aritméticas (+, -, x), de uso comum no processamento de sinais, são trocadas pelas operações algébricas ($\cap$ e $\cup$).

A não-linearidade que caracteriza a visão humana e a morfologia implica numa redução da informação a ser analisada em três fases:

1) redução do campo de investigação;
2) redução espectral que passa de uma imagem em níveis de cinza a uma imagem binária;
3) simplificação da imagem que aplica transformações morfológicas esquematizando a estrutura dos objetos a serem descritos.

No quadro da Morfologia Matemática, a visão de um objeto é percebida através da eliminação de todos os outros (Guichou, 1985). A metodologia consiste na supressão progressiva da informação irrelevante a partir de sequências de transformações.

A seguir serão apresentadas transformações fundamentais e exemplo de aplicação.

2 - TRANSFORMAÇÕES MORFOLÓGICAS BÁSICAS

Um tratamento morfológico vai geralmente envolver dezenas de transformações. Para domi

nar o processo é necessário conhecer suas propriedades que descrevem as relações entre o objeto inicial X e o objeto transformado $\psi(X)$:

propriedade 1: quando o objeto transformado $\psi(X)$ for incluído no objeto inicial, a transformação é chamada *extensiva* (antiextensiva, no caso contrário);

propriedade 2: o *crescimento* caracteriza as transformações que respeitam a hierarquia da inclusão, por analogia às funções crescentes:

ψ é crescente $\Longleftrightarrow X \subset Y \Longrightarrow \psi(X) \subset \psi(Y)$.

As transformações morfológicas são baseadas na ação de um elemento estruturante, ou seja, um conjunto de "pixels" ligados a uma origem O, que se comporta como uma sonda através da imagem E. A configuração-padrão do elemento estruturante é comparada com a configuração local. O valor no ponto central O é modificado em função da composição da vizinhança.

A imagem-teste (Figura 1) que foi escolhida representa um corte transversal de fibras de eucalipto com uma ampliação de 400 vezes. Na Figura 2 a imagem já está binarizada, a partir de um processo de "fatiamento" dos níveis de cinza.

2.1 - EROSÃO

No espaço euclidiano, a erosão denotada Θ elimina todas as partes do conjunto X que não podem incluir um elemento estruturante B.

2.1.1 - PROCEDIMENTO DE OBTENÇÃO

A erosão transforma todos os "pixels" do objeto, que não possuem a vizinhança padrão B, em fundo da imagem (Figura 3):

$$B_1 = \begin{matrix} 111 \\ 111 \\ 111 \end{matrix}, \text{ no caso de uma trama quadrada,}$$

$$B_2 = \begin{matrix} 1\ 1 \\ 1\ 1\ 1 \\ 1\ 1 \end{matrix}, \text{ no caso de uma trama hexagonal.}$$

2.1.2 - PROPRIEDADES

A erosão é crescente e antiextensiva, pois o conjunto $X \Theta B$ está incluído no conjunto X. A erosão é uma transformação que participa da eliminação dos artefatos da imagem. Existe uma propriedade muito interessante relativa à iteratividade:

$X \Theta nB = \underbrace{X \Theta B \Theta B \ldots \Theta B}_{n \text{ vezes}}$; onde n é o raio do elemento estruturante.

Ex.: B = vizinhança 3 x 3: $n = 1$,

B = vizinhança 5 x 5: $n = 2$.

Por exemplo, uma erosão por um elemento estruturante 5x5 pode ser obtida a partir de duas erosões consecutivas por elemento estruturante 3x3. É então possível fazer erosões de um elemento estruturante de um tamanho qualquer nB, a partir do elemento estruturante $B = 3 \times 3$.

2.2 - DILATAÇÃO

No espaço euclidiano, a dilatação denotada por $\oplus$ preenche os buracos (pontos "0") menores que o elemento estruturante B.

2.2.1 - PROCEDIMENTO DE OBTENÇÃO

A dilatação transforma em "1" os pontos do fundo que possuem algum "pixel" "1" na vizinhança deles (Figura 4).

2.2.2 - PROPRIEDADES

A dilatação é crescente e extensiva, i.e., $X \oplus B$ inclui o conjunto X. Ela é a transformação dual da erosão: a dilatação de um objeto X é o complemento da erosão do complemento X^c:

$$X \oplus B = (X^c \ominus B)^c.$$

2.3 - ABERTURA

A abertura do conjunto X, denotada X^B, consiste numa erosão seguida de uma dilatação, corta "cabos" e elimina "ilhas" menores que o elemento estruturante B (Figura 5).

A abertura é crescente, antiextensiva e *idempotente*. Esta última propriedade é um passo importante de uma sequência de transformações morfológicas, ou seja, a abertura realiza uma filtragem de partículas sem precisar de análise individual. Com sistemas especializados, esta operação é feita em tempo real. As aberturas com elemento estruturante de dimensão $1B$, $2B$, ... nB podem fornecer um método rápido para a obtenção de distribuições do tamanho das partículas (Serra, 1982).

2.4 - FECHAMENTO

Um fechamento, notado X_B, é a combinação de uma dilatação seguida de uma erosão. Esta operação é a dual da abertura. O efeito euclidiano é o preenchimento dos "lagos", "golfos" e "estreitos" menores que o elemento estruturante B(Figura 6).

As transformações apresentadas acima operam em paralelo em toda a cena. No processo de análise de imagens, o objetivo é geralmente a medição de características geométricas (superfície, perímetro, corda, etc.) relativas às partículas. Para isto, é necessário individualizar as componentes convexas. O terceiro capítulo enfoca a obtenção da fórmula de um dos possíveis parâmetros, isto é, o perímetro e seus algoritmos correspondentes no quadro da Morfologia Matemática.

3 - EXEMPLO DE MEDIDA: PERÍMETRO

3.1 - DEFINIÇÃO

O valor exato do perímetro $U(X)$ de um objeto X convexo é obtido a partir da integração da projeção $D(X,\alpha)$ sobre uma reta $\Delta\alpha$, α variando de 0 a 2π. O perímetro é o valor médio de $D(X,\alpha)$ multiplicado por π (relação de Cauchy):

$$U(X) = \pi \times \frac{1}{2\pi} \int_0^{2\pi} D(X,\alpha)d\alpha.$$

A fórmula de Crofton mostra que o valor do perímetro $U(X)$ pode ser aproximado usando um número de ângulos N, não-infinito, que varia de 0 a π:

$$U(X) \approx \pi \times \frac{1}{N} \sum_{n=1}^{N} D(X,\alpha n).$$

No caso digital, o valor da projeção $D(X,\alpha n)$ não é uma grandeza contínua, mas um múltiplo da distância de amostragem α.

3.2 - ALGORITMOS DO PERÍMETRO

Usando a fórmula de Crofton, a digitalização do perímetro resulta da discretização das projeções $D(X,\alpha)$ e do número limitado de retas de projeção de 0 a 2π:

1) Trama quadrada:
   A estimativa do perímetro é o valor médio das projeções $D(X,\alpha)$ sobre as retas $\alpha = 0$, $\alpha = 45$, $\alpha = 90$ e $\alpha = 135$. O perímetro é então obtido a partir da fórmula a seguir:

   $$U(X) = a[N(01) + N(\overset{1}{0})] + a\frac{\sqrt{2}}{2}[N(0^1) + N(^01)].$$

2) Trama hexagonal:
   Conforme a fórmula de Crofton já mencionada, o perímetro é deduzido a partir do valor médio das projeções sobre os três eixos principais do hexágono ($\alpha = 0^0$, 60^0, 120^0). A expressão do perímetro é:

   $$U(X) = \frac{\pi a}{2\sqrt{3}}[N(01) + N(0^1) + N(^10)].$$

3.3 - OUTRAS APROXIMAÇÕES DO PERÍMETRO

Estas aproximações são obtidas a partir das rotinas básicas de erosão e dilatação.

O perímetro externo pode ser considerado como a área da diferença do conjunto $X \oplus B$ menos o conjunto X, (B sendo o elemento estruturante padrão):

$$U_1(X) \approx [A(X \oplus B) - A(X)].$$

O perímetro interno é aproximado a partir da área da diferença do conjunto X menos o conjunto $X \ominus B$:

$$U_2(X) = A(X) - A[X \ominus B].$$

CONCLUSÕES

Foi apresentada uma abordagem original da análise de imagens binárias computadorizadas, a Morfologia Matemática. A filosofia geral consiste na transformação dos objetos da cena para facilitar a medição de parâmetros geométricos. As aplicações industriais encon

tram-se nas áreas das imagens microscópicas, de sensoriamento remoto e, ultimamente, em visão robótica, com as potencialidades da Morfologia em níveis de cinza.

A difusão da Morfologia Matemática, cujas bases foram desenvolvidas quinze anos atrás, foi lenta até o início da presente década por causa da não-adaptação dos processadores convencionais: por exemplo, uma erosão numa imagem de 512 x 512 pontos leva em torno de 2 minutos num microcomputador EGO. Atualmente, soluções por "hard" ou por "array processor" reduzem este tempo de processamento a um décimo ou a um vigésimo de segundo.

REFERÊNCIAS BIBLIOGRÁFICAS

BEUCHER, S. Watersheds of functions and picture segmentation. Colloque Image, Gretsi Cesta, Biarritz, 1984.

KENDALL, M.G.; MORAN, P.A.P. *Geometrical Probability*. London, Griffin, 1963.

GUICHOU, C. Análise de imagens e morfologia matemática. São José dos Campos, INPE, 1985 (no prelo).

MATHERON, G. *Random Sets and Integral Geometry*. New York, NY, Wiley, 1975.

PREUTEUX, F. Détection automatisée de differents organes en coupes scanner. Colloque Image, Gretsi, Cesta, Biarritz, 1984.

SANTALÓ, L.A. *Integral probability and geometric probability. Reading. MA*, Addison-Wesley, 1976.

SERRA, J. *Image Analysis and Mathematical Morphology*. London, Academic, 1982.

_____ Morphologie Mathématique et génèse des concrétions carbonatées des minerais de fer de Lorraine. *Sedimentology:* 183-208, 1968.

_____ Images et Morphologie Mathématique. *La Recherche, nº 144:* 727-748, 1983.

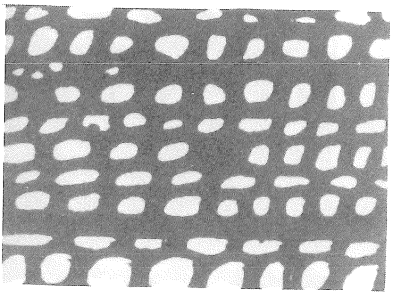


Fig. 1 - Corte transversal de fibras de eucalipto (400 x).

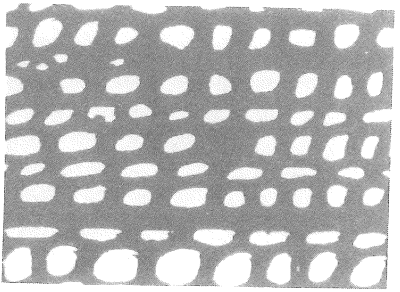


Fig. 2 - Binarização da imagem 1.

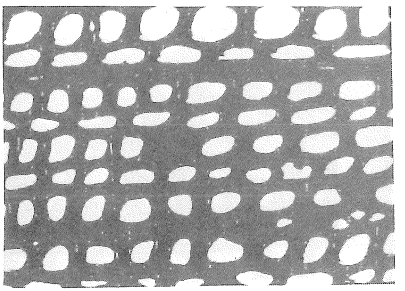


Fig. 3 - Erosão aplicada à imagem 2.

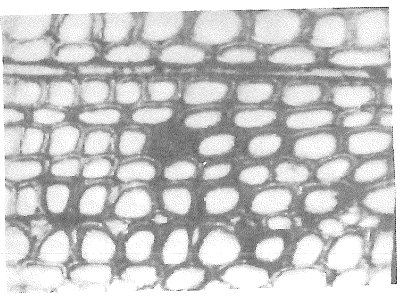


Fig. 4 - Abertura aplicada à imagem 2.

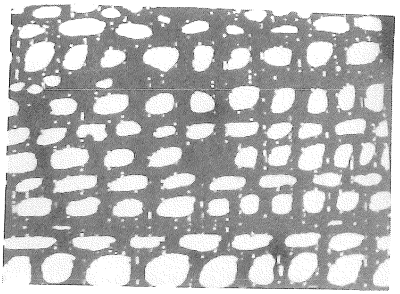


Fig. 5 - Dilatação da imagem 2.

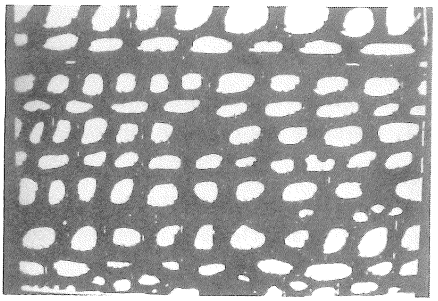


Fig. 6 - Fechamento da imagem 2.

G.P. AZEVEDO * J.A.S. BORGES **

SUMÁRIO

Este artigo descreve o terminal gráfico de média resolução que foi desenvolvido pelo NCE/UFRJ. Seu funcionamento como terminal TTY permite a qualquer computador o acesso a funções gráficas através de uma inteface RS-232C.
Suas funções gráficas são disponíveis através de um interpretador gráfico mnemônico, o que possibilita uma grande facilidade de programação gráfica em qualquer ambiente de programação.

ABSTRACT

This article describes the medium resolution Graphics Terminal designed at NCE/UFRJ. It works as a TTY terminal, therefore any computer may access its graphical resources by a RS-232C interface.

The Terminal's graphical functions are available through a mnemonic interpreter, which makes graphical programming easy and flexible in any enviroment.

* Graduando em Informática(UFRJ); Software Básico; Sistemas Operacionais;Compiladores; Estações para CAD; Universidade Federal do Rio de Janeiro, Núcleo de Computação Eletrônica, Caixa Postal 2324, Rio de Janeiro, CEP 20001.

** Informático(1980 - UFRJ); Processamento Gráfico; Estações de CAD; Universidade Federal do Rio de Janeiro, Núcleo de Computação Eletrônica, Caixa Postal 2324, Rio de Janeiro, CEP 20001.

INTRODUÇÃO

Este trabalho descreve a filosofia de funcionamento do terminal gráfico colorido de média resolução que foi desenvolvido no NCE/UFRJ.

Este terminal funciona com o protocolo TTY, logo pode ser ligado a qualquer computador através de uma interface RS-232C, no mesmo lugar onde seria ligado um terminal burro. Ele permite o acesso a funções gráficas independentemente do ambiente de programação e da existência de pacotes gráficos no computador em que estiver ligado.

Foram desenvolvidas duas versões do firware para hardwares distintos. Aqui é apresentada a versão para o hardware do Terminal Gráfico. A outra versão foi desenvolvida para o SDE-42 (micro fabricado pela EBC) com as placas de Vídeo-Gráfico, transformando um micro nacional em uma poderosa estação de trabalho remota CAD/CAM. As diferenças entre as versões são as inerentes as do hardware. Assim na versão Estação de Trabalho há funções para a transferência de dados, mas não são disponíveis as facilidades de interação proporcionadas por um teclado mais sofisticado e por um joy stick analógico.

MOTIVAÇÃO DO PROJETO

Desde a sua criação, o NCE/UFRJ vem oferecendo serviços gráficos, utilizando traçadores (plotters) ligados aos seus computadores. Entretanto, até 1980, essa atividade era basicamente de suporte aos usuários, produzindo programas de apoio e manutenção dos pacotes gráficos dos fabricantes.

Com a crescente necessidade de novos equipamentos gráficos dentro da instituição surgiu a idéia de criar-se uma área de pesquisa que fornecesse o "know how" de projeto desses equipamentos e do software básico. Vários fatores influiram de maneira direta nesta decisão:

- inexistência de equipamentos nacionais para gráficos por computador
- equipamentos importados são extremamente caros e quase impossíveis de conseguir.
- aumento da demanda de gráficos interativos pela comunidade acadêmica, especialmente o início de atividades de projeto assistido por computador (CAD).
- necessidade de gráficos on-line na área de controle de processos e equipamentos (CAM).

Em especial um outro projeto do NCE teve influência mais decisiva: o projeto de circuitos integrados VLSI. Um chip é composto de milhares de retângulos de silicio e metal organizados de tal forma que executem funções elétricas e lógicas extremamente complexas. O processo de desenho e montagem de partes é impossível de ser feito a mão pelo grande número de detalhes. Assim é necessário um equipamento especializado para desenho auxiliado por computador. Um equipamento importado custa entre 500 mil a um milhão de dólares. Nossa opção foi então projetar todo o conjunto de hardware e software que fosse capaz de atender ao projeto e que viriam, possivelmente a ser utilizados pelo resto

da universidade.

Dessa forma, criamos a partir de 1980 uma série de equipamentos e software, como o Terminal Gráfico (1984), que hoje estão a disposição da comunidade da UFRJ, alguns inclusive industrializados.

DESCRIÇÃO DO HARDWARE

O hardware do Terminal Gráfico é composto pelo Vídeo-Gráfico (Monitor gráfico colorido de rasteio a partir de uma TV comercial descrito em [Oliveira,82]); por uma UCP-Z80A ; por uma USART para comunicação com o computador e outra para impressora gráfica e por uma pastilha 8255 para controle do teclado, do joy stick, e dos leds.

O Vídeo-Gráfico tem duas telas, cada uma com 400 por 320 pontos. A primeira é usada como tela alfanumérica e gráfica (toda comunicação com o computador é feita através de - la), e a segunda exclusivamente como tela gráfica. Cada tela tem 320 linhas disponíveis para desenho, das quais é possivel selecionar como visíveis a um dado momento 240 linhas contíguas. O Vídeo-Gráfico dispõe de 16 cores diferentes que podem ter ainda 2 atributos: blink e cor predominante.

DESCRIÇÃO DO FIRMWARE

O firmware do Terminal Gráfico suporta três processos assíncronos e dois síncronos que gerenciam seus recursos e controlam seu funcionamento. Os processos assíncronos são ativados por interrupções e os síncronos estão ativos durante o resto do tempo. Os processos assíncronos são: os controladores de comunicação, do teclado e de hardcopy. Os síncronos são: o controlador de execução e o processo Idle que consome todo o tempo livre do sistema.

As estruturas de dados principais são as filas de execução, de transmissão e de hardcopy. Elas são implementadas por buffers circulares. A fila de execução contem dados produzidos pelo controlador de comunicações, durante a recepção de caracteres do computador, e pelo controlador de execução na expansão de macros, e tem como consumidor o controlador de execução. Os controladores do teclado e de execução são os produtores da fila de transmissão, cujo consumidor é o controlador de comunicações durante a trans - missão de caracteres ao computador. Já a fila de hardcopy tem como produtor o controlador de execução e como consumidor o controlador de hardcopy.

Cabe ao controlador de comunicações a transmissão e a recepção de caracteres do computador. Os caracteres a serem transmitidos saem da fila de transmissão e os recebidos entram na fila de execução. O controlador de comunicações faz parte do controle da linha de recepção, através do uso do protocolo XON/XOFF. Quando a fila de execução está 3/4 cheia e enviado um XON (control S) pedindo a suspensão do envio de caracteres pelo computador.

O controlador do teclado é acionado sempre que algo é teclado. Dependendo da tecla pressionada há um processamento local ou ela é convertida para ASCII e colocada na fila do buffer de transmissão.

O controlador de hardcopy é acionado quando o semáforo associado a sua fila está ligado, informando da existência de imagens a serem transmitidas e a impressora gráfica está pronta para receber mais uma parte da imagem.

O controlador de execução está ativo quando o semáforo associado a sua fila está ligado indicando que há caracteres a serem executados. Ele faz a outra parte do protocolo XON/XOFF, assim caso um XOFF tenha sido enviado, este só será anulado por um XON quando a fila de execução já tiver sido consumida de 3/4 de seu tamanho total. O controlador de execução consome os caracteres de sua fila, chamando o interpretador alfanumérico para interpretá-lo.

O interpretador alfanumérico analisa o caracter recebido e executa a função apropriada. Sua implementação é bastante simples, sendo baseada em estruturas de controle do tipo case. Entre as funções do interpretador alfanumérico estão as funções dos terminais tradicionais como BELL, HT, LF, FF, CR e as de endereçamento de cursor. Outras funções controlam a cor do traçado das letras, a cor do fundo, a cor do cursor, chamam os interpretadores gráficos e etc. As funções do interpretador alfanumérico são descritas no capítulo referente a parte alfanumérica.

O interpretador gráfico mnemonico é ativado pelo interpretador alfanumérico após o recebimento da sequência "Esc G", e é desativado por um CR, ou LF, ou ambos. Os caracteres recebidos quando o interpretador está ativo são considerados como comandos gráficos. Os comandos gráficos são formados por um mnemônico com duas letras, seguidas ou não por parâmetros numéricos. A identificação dos comandos é feita por hashing nos mnemônico.

Os principais algorítimos gráficos implementados; como o flood fill, boundary fill, desenho de caracteres, circunferências, retas e de símbolos se baseiam ou são encontrados em [Newmann 79], [Foley 82] e [Borges 83].

O firmware gráfico do Terminal dispõe de algumas características interessantes como:

- interpretador com comandos mnemônicos;
- interpretador binário não mnemônico para maior rapidez de traçado;
- possibilidade de edição local de gráficos com a entrada de comandos gráficos pelo teclado;
- modo trace no qual "Esc G" é mostrado na tecla como "$" e os comandos gráficos recebidos não são executados, mas listados;
- escolha da localização da origem;
- os parâmetros numericos podem ser números reais, sendo aproximados para o inteiro mais próximo;
- clipping automático: há uma checagem dos parâmetros das funções e somente é desenhado o que estiver dentro de uma área da tela abilitada pelo usuário. Quando tenta-se desenhar fora da tela o beep soa;
- os comandos gráficos só são executados se com parâmetros válidos;
- beep programável que avisa quando uma função recebe um parâmetro inválido.

A PARTE ALFANUMÉRICA

A tela alfanumérica tem 24 linhas de caracteres cada uma com 64 colunas. Os caracteres são desenhados numa matriz de 5 por 8. Caracteres vizinhos são separados por 1 ponto . As linhas de caracteres tem uma altura de 10 linhas, uma linha de separação da linha de caracteres anterior, 8 linhas para as letras e uma linha para o cursor. O cursor sublinha os caracteres. Os caracteres que tem uma "perninha" como f,g,j,p,q e y se utilizam sa oitava linha para serem desenhados corretamente.

A cor de traçado das letras é chamada de cor da letra, assim como a cor dos pontos não pertencentes a letra de cor do fundo. As cores da letra, do fundo e do cursor podem ser escolhidas indepentemente, podendo até ser transparente.

Entre as funções alfanuméricas implementadas estão as usuais (BEL, BS,HT,LF.FF,CR); as funções de movimentação e posicionamento do cursor, home, scrol reverso, apaga a frente e ao fim da linha. Além destas há funções para o acesso das características gráficas alfanuméricas como definir novas cores para o cursor, as letras e o fundo; para ativar os interpretadores gráficos mnemônico e binário e para trocar a tela sendo mostrada. É uma função para redefinir o modo de transmissão/recepção dos caracteres do computador.

PROGRAMAÇÃO GRÁFICA UTILIZANDO O TERMINAL

O Terminal Gráfico foi projetado para funcionar como um terminal com protocolo TTY.Desta forma ele pode ser ligado a qualquer computador, através de uma interface RS-232C , no mesmo lugar onde seria ligado um terminal burro. Pode, portanto, funcionar tão bem como console de um microcomputador de 8 ou 16 bits, quando ligado a um equipamento de grande porte. Esta característica aliada ao interpretador gráfico mnemônico, permite que no computador não haja necessidade de pacotes gráficos nem de modificações ou implementações no sistema operacional para se ter acesso às funções gráficas. É extremamente fácil o desenho diretamente através de linguagens de programação como C, PASCAL, BASIC, COBOL, FORTRAN, PL/1 ou DBASE-II.

O interpretador gráfico é ativado quando o terminal recebe a sequência "Esc G" e é ativado por um CR ou LF ou ambos. Os caracteres recebidos quando o interpretador está ativo são interpretados como comandos gráficos e são executados sequencialmente na ordem em que são recebidos.

Um comando gráfico é formado por uma sequência com um mnemônico, de duas letras, sucedido quando necessário por parâmetros numéricos. Cada mnemônico ou parâmetro numérico é considerado pelo interpretador como uma unidade básica independente, assim como uma palavra numa frase. As "palavras" podem estar entre um número qualquer de separadores (espaços, tabs ou vírgulas); sendo que entre parâmetros numéricos é necessário pelo menos um separador.

Podemos desenhar no Terminal Gráfico enviando-lhe um texto. Os parágrafos são formados pelos caracteres entre um "Esc G" e um CR ou LF ou ambos. As frases são os comandos gráficos e as palavras os mnemônicos e os parâmetros numéricos. Dentro da filosofia geral do Terminal Gráfico de facilitar seu uso nos mais diferentes ambientes de programação os mnemônicos podem ser formados indiferentemente por minúsculas ou maiúsculas.

Alguns conceitos são básicos para a programação do Terminal Gráfico, além de simplificar a sintaxe dos comandos e diminuir o número de parâmetros necessários para a maioria dos comandos, eles são:

- a tela gráfica : é a tela onde atuam todas as funções gráficas, ela é alterada com o comando "TG";
- a posição atual: é a posição de referência para o início do desenho das gráficas, ela é alterada pelo comando move "MV" e é também alterada de modo regular e previsível pelas funções gráficas, ela é relativa a origem.
- a cor atual: é a cor de trabalho no momento, equivalente a cor da pena em um plotter, ela só é alterada pelo comando "CO";

Se por exemplo quisessemos limpar a tela com a cor verde e desenhar duas circunferências amarelas no centro da tela em BASIC teríamos:

```
10 GRAF$  =  CHR$(27) + "G"
20 CIRCUN$ = "CI"
30 LIMPATELA$ = "LI"
40 NOVACOR$ = "CO"
50 MOVE$ = "MV"
60 PRINT GRAF$; LIMPATELA$,1; NOVACOR$; 2; MOVE$; 200, 120;
70 PRINT CIRCUN$; 40; CIRCUN$; 80
```

Assim como poderíamos ter feito:

```
10 PRINT CHR$(27);"G LI 1 CO 8 MV 200,120 CI 40 CI 80"
```

O mesmo programa em PASCAL poderia ser:

```
program TESTE (OUTPUT);
type
   COMGRAF = packed array [1..2] of char;
var
   INIGRAF,
   CIRCUN,
   LIMPATELA,
   NOVACOR,
   MOVE:  COMGRAF;
   X, Y: integer;
begin
   INIGRAF [1] : = chr(27) ;
```

```
    INIGRAF[2] : = "G" ;
    CIRCUN : = "CI" ;
    LIMPATELA : = "LI" ;
    NOVACOR : = "CO" ;
    MOVE : = "MV" ;

    X : = 200 ;
    Y : = 120 ;

write (INIGRAF, LIMPATELA, "1");
write (NOVACOR, "2", MOVE, X, Y) ;
writeln (CIRCUN, "40", CIRCUN, "80") ;
end.
```

USO DO TERMINAL GRÁFICO NO NCE/UFRJ

O terminal está funcionando desde Abril/84, ligado a vários computadores: Micros de 8 bits(com sistema operacional CP/M), um PDP-11/70 (com Unix), um Burrough's B6700 e ao PEGASUS 32X, supermicro sendo desenvolvido pelo NCE em conjunto com o sistema operacional PLURIX.

Ele já foi exposto ligado ao PEGASUS 32X na IV Feira Internacional de Informática, e ao PDP-11/70 durante o 3º Semicro.

FUTURO DOS TERMINAIS DE VÍDEO NO BRASIL

Pode-se prever que num futuro breve não haverá mais lugar para os terminais puramente alfanuméricos no mundo. O custo da memória, que é o elemento mais crítico dos terminais , baixa continuamente e todos os novos controladores de vídeo possuem possibilidade de geração gráfica.

O problema do projeto de terminais no Brasil é a falta de tubos de imagem com capacidade alta, o que nos limita a produção de terminais com alta resolução. Porém existe um mercado relativamente grande que pode ser atingido com terminais de resolução média. Urge a criação de equipes com capacitação para projetar software gráfico, tanto básico (firmware de terminais) quanto aplicativo.

BIBLIOGRAFIA

Newmann 79: Newmann & Sproull - PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS
MacGraw Hill - 1979

Foley 82 : Foley & Van Dan - FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS
Addison-Wesley - 1982

Borges 83 : Borges, J.A. - SOFTWARE BÁSICO PARA TERMINAIS GRÁFICOS
Relatório Técnico NCE-0183 - NCE/UFRJ

Oliveira 82: Oliveira, C.E.T. - UM MONITOR GRÁFICO A CORES
II CSBS - Anais do IX SEMISH

J. ZAMBRANO *

RESUMEN

El siguiente trabajo presenta los elementos a ser considerados en la realización de diálogos (interactivos) para la enseñanza de la programación, siguiendo como metodología la realización de maquetas experimentales que verifiquen las hipótesis de trabajo provenientes de una especificación pedagógica. Estas maquetas están concebidas como un recurso suplementario en una iniciación a la programación y ellas responden a varios objetivos: validar un enfoque de la enseñanza de la algoritmica e integrar la herramienta en este enfoque, validar una manera de trabajo informática de concepción y de producción de programas didácticos.

ABSTRACT

Key elements to be considered during performance of C.A.I. dialogues are discussed in this paper. The mothodology herein followed rests upon the building up of experimental models in order to verify set of hypotheses attached to particular work issues of a pedagogic nature.

Such a progressive family of experimental models is conceived as a complementary aid in programming learning and several objetives are aimed to, mainly to asses:

- a teaching programming approach incorporating the tool we present here
- a computational practice for the Computer Assisted Software production.

* Licenciado en Computación (Vls. 1975)
Doctor en Informática (FRA, 1984).
Universidad Central de Venezuela, U.C.V.
Facultad de Ciencias, Escuela de Computación,
Apartado 47002, Caracas 1041-A. Venezuela.
Tlf. (02) 6627233

1. INTRODUCCION

Abordamos en este trabajo algunos aspectos de la enseñanza de la programación que sigue los principios que han sido acordados por la comunidad informática actual. Una enseñanza tal debe considerar tanto los métodos propios a la construcción como las técnicas que permiten la descripción de programas, tomando en cuenta las restricciones impuestas por el medio informático utilizado y la calidad que se espera del programa-producto.

Dentro de este contexto se busca ubicar entonces a la programación como una actividad que abarque las etapas siguientes: elaboración de un cuaderno de notas (conocimiento del problema), análisis del problema que conduzca a una especificación en términos informáticos, construcción de un programa, pruebas, redacción de documentos, integración en el sitio de utilización, explotación y mantenimiento.

Podemos resumir acá, algunos elementos fundamentales sobre la actividad de la programación que deben aclarar la concepción de su enseñanza:

a) La actividad de la programación reposa en una clara distinción entre el análisis, la construcción propiamente dicha y la verificación progresiva de la corrección.

b) El programa-producto, es decir el resultado del trabajo del programador, debe ser apreciado en términos de diversos factores: fiabilidad, seguridad, robustez, adaptabilidad, y mantenimiento. Además el programa producto debe consistir no solamente de un texto final redactado en un lenguaje de programación sino también de diversos documentos necesarios para su explotación y su mantenimiento: documento de análisis, documento técnico y documento para el usuario.

Esta visión sobre la actividad de la programación, nos permite analizar algunos problemas específicos a la enseñanza de la programación y justificar entonces la búsqueda de soluciones que integre al computador para ayudar a esta enseñanza.

Estos problemas específicos pueden ser resumidos al considerar de manera general los aspectos siguientes: el público a quien va dirigida la enseñanza (diferentes edades, diferentes profesiones o actividades, carencia en general de una cultura informática, problemas de masa estudiantil...), la importancia del trabajo conceptual en la actividad de programación que se opone constantemente a los aspectos técnicos de la disciplina y a las relaciones psicológicas particulares del público desde el punto de vista de las máquinas (aspectos lúcidos, comportamientos artisanales,...). Se debe en este sentido recordar algunos mecanismos intelectuales que condiciona la facilidad con la cual el estudiante podrá asimilar el fundamento de la programación: capacidad de abstracción, capacidad de analogía y aptitud al razonamiento formal.

La importancia de esos mecanismos intelectuales aparece desde el comienzo del aprendizaje de la programación: se deben introducir simultáneamente las nociones de proceso, algoritmo, y análisis descendente, así como las notaciones para expresarlas.

Sobre la base de estas observaciones que acabamos de hacer sobre la programación y los pro

blemas impuestos para su enseñanza, hemos podido discernir donde se podría realizar un aporte de la enseñanza asistida por computador en una enseñanza tal. La existencia de un cierto consenso sobre la enseñanza de la algoritmica permite vislumbrar la concep - ción de programas didácticos en el tema.

Enseñanza asistida por computador -E.A.C.-

Queremos situar nuestro trabajo sobre la base de un análisis de los problemas que se im ponen actualmente en E.A.C.

En un principio la E.A.C. estaba considerada como una solución a problemas pedagógicos propios de una minoría, por lo que se aceptaba (por razones políticas o sociales) cos - tos relativamente importantes: retardos escolares de grupos socioculturales particula - res, formas de enseñanza destinadas a minusválidos, etc.. Esta problemática ha sido to- talmente transformada con la llegada de la microinformática, la telemática y la informá tica individual (BeF-82).

La E.A.C. se ha convertido ahora en un fenómeno de masa sometido al impacto de los me - dios, en una sociedad donde la informática por una parte y los problemas de la educación por otra, ocupan un lugar predominante.

Lo que caracteriza a la E.A.C. en el sistema educativo, es un esfuerzo centralizado de equipamiento de los establecimientos, de formación de los docentes, de desarrollo de he rramientas y de normalización de esas herramientas. El objetivo de esta informatización es, a corto plazo, hacer efectiva la integración de la informática en la pedagogía apo- yándose en una participación activa de los docentes.

Aunque los objetivos y enfoques de la E.A.C. son muy variados, se destaca una orienta - ción tendiente a transformar los sistemas educativos actuales sobre la base de un prin- cipio de educación individualizada gracias al computador (Pap-83); esto ha dado origen a diversas tendencias en la forma de trabajar en E.A.C.:

- lenguajes especializados (lenguajes-autores)
- el enfoque del "niño programador" (Pap-80), (Pap-83)
- productos terminados por casas editoras o sociedades de servicio especializadas
- una práctica general que integre los diversos aspectos de la concepción, de la produc ción, de la explotación y de la evaluación de programas didácticos.

Presentamos en este trabajo los elementos básicos (pedagógicos e informáticos) que deben ser tomados en cuenta para la realización de maquetas de trabajo que avalen ciertas hipó tesis sobre la enseñanza de un sujeto y proponemos su realización de manera sistemática que permitan por una parte la verificación de las hipótesis y por la otra la integración de funciones pedagógicas provenientes del análisis pedagógico realizado que incrementen las soluciones informáticas utilizadas en un contexto de E.A.C.

La metodología consiste entonces, en una clara separación de las fases de análisis peda- gógico y de realización informática. Esta separación permite no solo la realización de un

análisis pedagógico más claro y sistemático sino además de discernir sobre las eventuales funciones pedagógicas que deben integrarse al contexto de E.A.C. Por otra parte se busca la definición de hipótesis de trabajo para la realización de nuevas maquetas y la integración de los resultados provenientes de la realización de esas maquetas para la producción de programas didácticos completos.

2. ESPECIFICACION PEDAGOGICA DE LA MAQUETA

La maqueta realizada es un apoyo a la acción tanto del alumno como del docente. Este apoyo debe permitir al alumno entre otras cosas: adquirir los conocimientos de forma individual, trabajar el tiempo que desee para disipar las dudas potenciales del curso correspondiente, concentrarse en un sujeto; en cuanto al docente, ella debe permitirle presentar aquello que es difícil a mostrar en un curso tradicional.

La elaboración de la maqueta se apoya en la escogencia de los elementos siguientes:

- Los sujetos tratados en el programa didáctico
- Los momentos pedagógicos del aprendizaje de estos sujetos
- Los enfoques pedagógicos utilizados para enseñar esos sujetos.

2.1 Sujetos abordados en la maqueta

La maqueta presentada ha sido concebida como apoyo a una iniciación a la programación, donde el estudiante deberá completar, al utilizar los programas didácticos, todo cuanto es desarrollado en clases por el docente. Para nuestra primera experiencia hemos seleccionado dos temas por considerarlos particularmente bien adaptados a una experiencia de E.A.C.: la noción de algoritmo y la noción de análisis descendente.

Consideraremos un documento de análisis como la descripción de un algoritmo cuyos componentes deben mostrar los diferentes niveles de abstracción del proceso seguido para obtener el programa correspondiente.

2.2 Modelos de algoritmos que sirven de soporte a la exposición de los sujetos escogidos.

Los algoritmos propuestos en la maqueta ilustran esencialmente la composición condicional y la composición iterativa para los dos tipos de ejercicios presentados.

Dos métodos de análisis se proponen a los estudiantes para las composiciones precedentes, que tienden a resolver la dificultad de comprensión de los sujetos: el análisis por casos y el tratamiento secuencial.

El análisis por casos: La ejecución de acciones en un proceso depende a menudo del valor de ciertas variables que determinan el estado corriente del proceso.

El análisis por caso ayuda a construir una solución analizando cada uno de los casos posibles determinados por el problema a resolver, y asociando a cada caso una acción que exprese la solución. Esta herramienta de análisis, impone un rigor sistemático que obliga a: precisar la lista de casos posibles y expresarlos; precisar las acciones que resuelven el problema y establecer una exclusión mutua entre los casos definidos.

El análisis por caso define entonces una partición del dominio de los datos donde se ha-

cen corresponder los casos y las acciones a fin de resolver el problema. Para el análisis por casos, nos apoyamos en las composiciones selectivas y condicionales (LPS-83) que hace precisar el conjunto de casos posibles y las acciones correspondientes.

El tratamiento secuencial: Este método (Sch-79), (LPS-83) utiliza un modelo abstracto de "máquina secuencial" que permite describir los esquemas de tratamiento secuencial haciendo abstracción de la representación de la secuencia dada.

El tratamiento secuencial de la información hace intervenir una máquina secuencial de acceso y una máquina secuencial de tratamiento.

El modelo nos permite hacer la manipulación de los algoritmos de la manera más simple para ilustrar las ideas más importantes de los sujetos en cuestión.

Una máquina de acceso secuencial permite acceder secuencialmente los elementos de una secuencia que se encuentra en una cinta.

La máquina vista desde el exterior presenta:

- una ventana llamada "CARACTER CORRIENTE" (CC)
- dos botones llamados "ARRANCAR" y AVANZAR" (ARR y AV)

El estado de la máquina está compuesto por:

- un indicador de secuencia vacía
- un elemento corriente
- un indicador de fin de secuencia

El repertorio de primitivas de la máquina de acceso (PeS-82), (LPS-83) está compuesto por las acciones de modificaciones y consulta siguientes:

Modificación

ARRANCAR (ARR): Da acceso al primer elemento de la secuencia
Este primer elemento es accesible por la ventana.

AVANZAR (AV) : Da acceso al próximo elemento de la secuencia.
Este elemento es accesible por la ventana.

Consulta

ELEMENTO CORRIENTE (EC) : Da acceso al elemento presente en la ventana.

FIN DE SECUENCIA (FDS) : Indica si se alcanzó el fin de la secuencia.

SECUENCIA VACIA (S_VACIA) : Indica si la secuencia está vacía

El repertorio de la máquina de tratamiento está compuesto de las acciones siguientes:

VISITAR : Es la acción elemental aplicada a todos los elementos de la secuencia

INICIALIZACION DE LAS VISITAS (INIC_V): Acción aplicada como prólogo del tratamiento secuencial.

TRATAMIENTO SECUENCIA VACIA (T_VACIO): Tratamiento para el caso en que la secuencia es vacía.

Un caso particular es la máquina contador (Zam-84) que representa un contador cuyo valor se muestra en la ventana.

La combinación de los repertorios de esas máquinas, según las características de la secuencia de información tratada, nos ha permitido definir un conjunto de esquemas (Moi-81), (PeS-82), (Zam-81). Estos esquemas se apoyan en las dos formas de composiciones iterativas (repetir y mientras).

A continuación se muestra el esquema utilizado en la maqueta, que permite el recorrido de una secuencia provista de una marca que caracteriza el fin de la misma. Una fila es vacía si no lleva más que la marca.

* Esquema de tratamiento secuencial

```
ARR
selección
   FDS  :  T_VACIO
   FDS  :  INIC_V
           repetir
             VISITAR (EC)
             AV
           hasta FDS
           TERM-V
fselección
```

Todos los esquemas están expresados con la ayuda de la notación algoritmica MEFIA (Mor-79), (Sch-79).

La selección es la herramienta algoritmica de la notación para expresar el análisis por caso.

3. LOS MOMENTOS PEDAGOGICOS

Nos hemos apoyado de manera clásica, en tres momentos pedagógicos, que corresponden a objetivos diferentes pero que su puesta en práctica hace intervenir medios comunes. Esos momentos son: la sensibilización, la presentación y la consolidación.

En la progresión pedagógica, cada momento está dedicado a un aspecto diferente y presenta también distintos niveles de dificultades.

3.1 Sensibilización al contexto de trabajo.

En este momento pedagógico, se introduce la categoría de problemas con los cuales se desea trabajar con el estudiante. Nos esforzamos en apoyarnos en los conocimientos del alumno y buscamos a través de la sensibilización, crear un interés por el tema llamando siempre la atención del aprendiz sobre las ideas de base que él debe dominar.

Para cada uno de los sujetos que han sido seleccionados, la sensibilización está asegurada a dos niveles. Se distinguen las actividades que se hacen en clases con el docente y aquellas que son el objeto de la E.A.C.

La sensibilización de la noción de algoritmo, por ejemplo consiste en hacer participar al estudiante en la elaboración de una sucesión de acciones que describan un proceso, conocido o deducido por él, a partir del enunciado de un problema (un juego, por ejemplo).

Se confronta al estudiante con situaciones que requieren de él una reacción en función de las nociones sobre las que se trabaja.

Los objetivos de la sensibilización están en relación con:

- La noción de acción: como interpretar un problema en términos de una sucesión de acciones. Cuál es la especificación de cada acción. (Toda acción debe ser realizada a priori).
- La notación algoritmica: Como expresar cada acción que describe el proceso de manera precisa y clara y como describir los objetos manipulados por cada acción.
- La noción de ejecución: Cual es el efecto producido por cada acción y por el conjunto completo de acciones.

En nuestra maqueta, nos beneficiamos de la sensibilización hecha en clases y utilizamos la E.A.C. para poner al estudiante en relación, de manera más concreta, con la categoría de problemas. El estudiante gracias a un sitio de trabajo que se le ofrece, puede manipular un conjunto de teclas-funciones, que representan las acciones que se desea hacerle comprender al mismo tiempo que él puede observar el efecto producido por cada acción así como el orden de ejecución correspondiente.

La sensibilización al análisis descendente tiene por finalidad poner al estudiante en relación con la concretización de un (o varios) nivel(es) de abstracción presente(s) en un algoritmo. Esta concretización mostrará, en particular, el pasaje de un nivel corriente a un nivel más bajo y también la descripción de las acciones que traducen ese cambio.

Se busca a inducir en el estudiante esta forma de trabajo para los diferentes niveles de abstracción de un algoritmo.

Los objetivos de la sensibilización están relacionados con:

- La descripción de un nivel de abstracción. (Pasaje de "qué hacer" a "cómo hacer").
- La aplicación de la misma gestión a todos los niveles.
- La integración de las composiciones algoritmicas en la expresión de un nivel del análisis.

Se confronta al estudiante con los diferentes niveles que surgen de un análisis que lleva a una solución. Estos niveles están representados, en la maqueta, por visualizaciones con la ayuda de encuadramientos en el algoritmo correspondiente afichado en una parte de la pantalla y la descripción del conjunto de acciones que traducen el cuadro en otra parte de la pantalla.

3.2 Presentación de conocimientos

Se presentan al estudiante los elementos fundamentales del sujeto abordado y sus relacio

nes. En nuestro contexto se presentan al alumno, modelos informáticos para resolver los problemas en los que ha sido sensibilizado.

Para la noción de algoritmo: la notación algoritmica utilizada; el análisis que establece la relación entre el enunciado y el algoritmo; la ejecución del algoritmo que establece la relación entre el algoritmo y la máquina.

Para el análisis descendente: el estudio de la estructura del algoritmo; el estudio de los componentes del algoritmo, en diversos niveles de abstracción.

El medio ambiente de E.A.C. es útil porque permite al estudiante poner en evidencia, a través de sus actividades, las relaciones existentes entre el algoritmo propiamente dicho del problema que se quiere resolver y la ejecución del proceso relacionado. El estudiante tiene mucha dificultad para comprender la relación existente entre un algoritmo (representación estática) y su ejecución (proceso dinámico). Otra dificultad se presenta para el estudiante con la noción de estado y sus cambios (un algortimo cambia el valor de las variables). Aunque se entiende como un algoritmo manipula datos particulares para producir un resultado, la comprensión de lo que pasa cuando se generaliza a los otros datos no siempre es evidente para el estudiante.

En cuanto al análisis descendente se intenta visualizar la estructura de un algoritmo. Se quiere mostrar al estudiante que la estructura de un algoritmo puede ser considerada bajo dos ángulos complementarios: entender como se construyó el algoritmo y entender como el algoritmo será ejecutado.

En resumen, la presentación de un algoritmo tiende a desarrollar en el estudiante las aptitudes siguientes:

- Capacidad de reconocer los componentes de un algoritmo dado.
- Capacidad de reconocer las relaciones entre esos componentes. (Nivel de abstracción, enlace entre estructura del programa y estructura de datos).

3.3 Consolidación de los conocimientos

Se busca en este momento pedagógico:

- Fijar los conocimientos presentados
- Fijar las relaciones con otros conocimientos adquiridos
- Ofrecer un contexto de experimentación para evaluar los conocimientos adquiridos.

Se distinguen dos objetivos precisos: de una parte consolidar los conocimientos que se presentaron al estudiante y por otra parte verificar si éste las ha entendido.

4. LOS ENFOQUES PEDAGOGICOS

Los enfoques pedagógicos permiten abordar los sujetos de manera particular (APS-81). Para enseñar los diversos sujetos hemos trabajado sobre la base de dos enfoques pedagógicos: la simulación y la lectura de un documento de análisis.

Es importante integrar al estudiante en su propio proceso de aprendizaje. Es a través de

la participación activa del alumno que se tiene más posibilidad de lograr esta integración.

En la maqueta hacemos participar al alumno activamente a través de los diferentes enfoques utilizados a los que hemos hecho corresponder herramientas de E.A.C. particulares.

Para la simulación disponemos de la máquina caracteres y la máquina contador, mientras que para la lectura disponemos de un mecanismo que nos permite tener acceso a los diferentes niveles de un algoritmo. Cada nivel muestra los componentes según un modo de lectura previamente definido.

La lectura de un documento de análisis por el alumno consiste por una parte a recorrer su estructura y por la otra a decifrar cada componente de esta estructura. Ella permite también percibir las relaciones existentes entre los diferentes componentes de un algoritmo, así como los diferentes niveles de abstracción que están presentes, según los objetivos del docente. Esto implica diferentes manera de leer un documento. Distinguimos tres tipos de lectura, según se quiere trabajar en la construcción de un algoritmo, la identificación de los componentes en el algoritmo o la semántica de éste.

En cuanto a la simulación, este enfoque pone a la disposición del alumno ciertas herramientas que podrá utilizar y que le permitirán obtener su propia enseñanza.

Estas herramientas corresponden a modelos simulados en el computador que están directamente en relación con una categoría particular de problemas. Así, por ejemplo, un modelo simulado de una balanza permitirá al estudiante, deducir de su propia actividad la acción a seguir para los problemas que pongan en evidencia las relaciones de orden.

Este enfoque está particularmente adaptado al momento pedagógico de sensibilización. Ella es completamente compatible con una pedagogía activa. En nuestro contexto, la simulación permite responder a las necesidades de los estudiantes, de concretizar los procesos dinámicos implicados por un algoritmo, controlando al mismo tiempo su actividad con la ayuda de modelos de máquinas coherentes con el problema metodológico que caracteriza la iniciación a la programación.

5. LAS SOLUCIONES E.A.C.

Visto los sujetos a tratar, la caracterización de los momentos pedagógicos y los enfoques pedagógicos para abordar esos sujetos, resta abordar el aspecto como utilizar esos elementos en la construcción de una solución E.A.C.. La elaboración de esta solución depende de la escogencia de una estrategia pedagógica y de los diferentes medios E.A.C. disponibles.

La estrategia para la definición de una solución E.A.C. se basa en: la presencia del enseñante; la progresión en la enseñanza, caracterizada por los tres momentos pedagógicos; la libertad dada al alumno y los enfoques pedagógicos que particularizan el aporte de la E.A.C.

La escogencia de los diferentes medios E.A.C. disponibles se apoya, por su parte, en un

conjunto de funciones pedagógicas que aparecen en curso del análisis pedagógico. Este permite traducir esas funciones en términos de funciones disponibles en el puesto de trabajo del alumno.

Entre los medios escogidos encontramos:

- Una organización particular de la pantalla para cada actividad.
- El encuadramiento, los atributos de aspecto y la animación que permiten: la designación de elementos, la puesta en relación de componentes y la gestión de diálogos.
- El sonido: refuerzo de fases importantes del diálogo.
- La definición de teclas-funciones; base para la actividad del alumno.

6. CONCLUSIONES

Aún cuando no describimos las soluciones E.A.C. específicas que hemos implementado en la maqueta, es importante destacar los elementos básicos de una manera de trabajar sistemática para la realización de programas didácticos a partir de un análisis pedagógico previo.

La especificación informática de las lecciones a realizar con un programa didáctico se apoya en una tipología precisa de las diversas funciones informáticas necesarias. Estas se deducen de las funciones pedagógicas provenientes del análisis, teniendo en cuenta el medio ambiente disponible.

Una producción que considere los elementos que hemos descrito deberá estar basada en una neta distinción entre el análisis pedagógico y la realización informática y además en el reconocimiento explícito de las competencias propias a los diferentes participantes de un proceso E.A.C. (aspecto pedagógico, aspecto informático).

El análisis pedagógico comprende el análisis del contenido, la definición de los momentos pedagógicos y la escogencia de los enfoques pedagógicos. Este análisis se funda, por una parte, en las experiencias pedagógicas y por la otra en una estrategia precisa de integración de la E.A.C. en el proceso de aprendizaje.

7. BIBLIOGRAFIA

(APS-81) ABRY, D; PAINVIN, S SCHOLL, M.
Un didacticiel pour l'eseignement du francais langue étrangére.
Rapport de spécificacion pédagogique
IMPG - IMAG - ULLG - CUEF, Francia 1981

(BeF-82) BESTOUGEFF, H FARGETTE, J.P.
Enseignement et Ordinateur
Cedic 1982

(LPS-83) LUCAS, M. PEYRIN, J.P. SCHOLL, P.C.
Algorithmes et représentation des donées:
Files, Automates d'etats finis
Masson 1983

(Moi-84) MOITA MONTE H.
Thése de docteur ingenieur
USMG INPG (en preparation) 1984

(Mor-79) MORAT, P.
Construction méthodique de programmes: MEFIA.
Etude de quelques primitives de controle
Rapport D.E.A. - INPG - Grenoble - 1979

(Pap-8Ø) PAPERT, S.
Computer and learning
The Computer Age MIT. Press - 198Ø

(Pap-83) -
Mindstorms: Children, Computer and Powerful ideas
Havester Press, Brighton - 1983

(PeS-82) PEYRIN, J.P. SCHOLL, P.C.
Méthodes de Programmation
USMG - Institut de Programmation - 1982

(Sch-79) SCHOLL, P.C.
Vers une programmation systematique: Etude de quelques
méthodes, techniques et outils.
Thêse d'etat - Grenoble - 1979.

(Zam-81) ZAMBRANO, J.
Sur un système expérimental de construction de
programmes.
Rapport D.E.A.
USMG - INPG 1981

(Zam-84)
E.A.O. et Enseignement de la programmation; Une maquette
de didacticiel. Thêse de docteur: 3eme Cycle
INPG - USMG Grenoble, Francia 1984

ESTUDO COMPARATIVO DAS DISCIPLINAS EXAUSTIVA E NÃO-EXAUSTIVA NA INTERLIGAÇÃO DE REDES LOCAIS *

L. LELLIS* E D.A. MENASCÉ**

SUMÁRIO

Em trabalhos anteriores foi demonstrado pelos autores que, a interligação de um grande número de estações, através de um conjunto de redes locais (LANs) interligadas pode ser vantajosa, do ponto de vista de performance, quando comparada com a solução que utiliza apenas uma rede.

Este artigo explora este assunto em mais detalhe. A influencia, no retardo fim-a-fim, do uso da disciplina exaustiva versus a não-exaustiva, é estudada. O impacto do uso da aproximação de carga balanceada, para a taxa de chegada no gateway e também verificada.

ABSTRACT

It has been demonstrated in previous works by the authors that the interconnection of a large number of stations through a set of interconnected local area networks (LANs) may be of advantage performance wise, when compared with a single LAN solution.

This paper explores this subject in more details. The influence, on end-to-end delay, of the use of exaustive versus non-exaustive disciplines is studied. The impact of the use of the balanced load approximation for the gateway arrival rate is also verified here.

* Analista de Sistemas da LIGHT S.E.S.A.

** Professor associado do Departamento de Informática da PUC/RJ, 22453 Rio de Janeiro-RJ.

Esta pesquisa foi parcialmente financiada pela FINEP e EMBRATEL (Contrato C.DCD-002/82).

1 - INTRODUÇÃO

Estudos recentes (LELL84,MENA84), demonstraram que particionar uma rede local, que possua muitas estações, em um conjunto de redes locais interligadas, pode melhorar bastante o desempenho nas comunicações entre as estações.

Esta melhoria se deve basicamente ao fato que, com o particionamento, cada rede passa a ter um numero menor de estações. Como o tempo de retardo aumenta não linearmente com o número de estações, esta diminuição do número de estações faz com que o retardo em cada rede seja bem menor que o retardo na rede unica, compensando e deixando uma margem de ganho sobre o tempo gasto na interligação das redes (tempo de retardo nos gateways).

Para analisar este efeito em (LELL84), foi desenvolvido um modelo aproximado, para um conjunto de redes locais interligadas, cada uma operando por passagem de permissão.

Em (MENA84) foi obtido um modelo analítico que fornece o retardo médio fim-a-fim entre duas estações do conjunto de redes interligadas.

Este modelo utiliza-se da aproximação de que a carga gerada pelos diversos gateways em uma dada rede será adicionada a carga gerada pelas estações usuárias e dividida pelo número total de estações (incluindo os gateways). Para o cálculo do tempo medio de espera em uma rede isolada, operando com cargas balanceadas e disciplina de atendimento não exaustiva foram utilizados os resultados de (LELL83b).

A figura 1 apresenta uma comparação entre os resultados obtidos através do referido modelo e de um modelo de simulação, para a curva retardo versus número de redes interligadas, utilizando a topologia em ANEL para interligação das redes, para um número total de 320 estações, com uma taxa de chegada média de mensagens de 10 mensagens por segundo (POISSON), sendo o tamanho médio das mensagens 1600 bits (exponencialmente distribuído) e uma capacidade de transmissão na barra de 10Mbits por segundo.

A tabela 1 complementa estes dados apresentando, além dos resultados analíticos e de simulação, o erro causado pela aproximação e as taxas de chegada reais nos gateways e nas estações usuárias, assim como sua aproximação.

NUM. REDES	PROCES. P/REDE	RETARDO			TAXA DE CHEGADA		
		ANALIT.	SIMUL.	ERRO	PROC. USUÁRIO	SAÍDA GATEWAY	APROX. BALANC.
2	160	2409	15125	84%	10	400	15
4	80	1969	2101	5%	10	400	22
8	40	2038	1930	6%	10	400	32
16	20	2708	2532	7%	10	400	49
32	10	4340	4440	2%	10	400	79

A análise destes resultados indica que:

i) a referida aproximação para a taxa de chegada balanceada é razoável, mesmo para os casos em que existe uma diferença entre a taxa de chegada gerada pelos gateways e estações usuárias (note o caso de 4 redes com 80 processadores, onde a taxa real nas estações usuárias é de 10 msg/seg, no gateway 400 msg/seg, a aproximação é de 22 msg/seg e o erro é da ordem de 5%).

ii) esta aproximação não é razoável, para o caso em que cada rede tem muitos processadores, pois subestima a sobrecarga introduzida no tempo de retardo de cada mensagem, que atravessa os gateways, pela passagem da permissão entre os diversos processadores de cada rede (note que para 2 redes com 160 processadores, o erro é de 84%, enquanto que para os outros casos, é inferior a 10%).

A análise destes resultados indica que para os casos em que se pretenda ter um pequeno número de redes interligadas, cada uma contendo muitos processadores, a disciplina de atendimento não exaustiva não é a mais conveniente, pois introduz uma sobrecarga muito grande para cada mensagem que atravessa os gateways. É interessante observar que este é um caso prático de grande interesse, visto que o mais natural em uma empresa, será iniciar com um pequeno número de redes interligadas e depois, este número ir aumentando.

Desta análise, surge a percepção de que a disciplina de atendimento exaustivo deve ser mais conveniente para os casos mencionados do que a disciplina não exaustiva.

Na seção seguinte, são feitas adaptações ao modelo apresentado em

(MENA84), visando avaliar o efeito da utilização da disciplina de atendimento exaustiva em lugar da não exaustiva, no retardo médio fim-a-fim em um conjunto de redes locais interligadas, operando com o protocolo de passagem de permissão em barra comum.

2 - MODELO MATEMÁTICO

PARÂMETROS DE ENTRADA

P : número de estações no conjunto de redes interligadas
R : número de redes interligadas
Ps,t: probabilidade que um pacote que chegue na estação 's' vá para a estação 't'
ls : probabilidade que um pacote externo chegue na estação 's'
G : número de gateways que interligam o conjunto de redes
Ri : conjunto de estações da rede 'i'
Si : conjunto de gateways da rede 'i'

De (MENA84), obtemos a seguinte expressão para o retardo médio fim-a-fim no conjunto de redes interligadas:

$$D = \sum_{i=1}^{R} \sum_{s \epsilon Ri} ls) \sum_{j=1}^{R} Dij \; Tij \qquad (1)$$

Onde, Dij é definido como sendo o retardo medio no envio de um pacote da rede i para a rede j e Tij é a probabilidade que um pacote que chegue na rede i vá para a rede j.

A obtenção de Dij leva em consideração o retardo em cada rede (Dii) e em cada gateway (DGg) e o caminho da rede i para rede j, inclusive os tempos nas redes i e j.

Assim sendo, a disciplina de atendimento (EXAUSTIVA OU NÃO EXAUSTIVA) influencia o retardo fim-a-fim através da influência que faz no retardo Dii em cada rede.

Em (LELL83b), é desenvolvido um modelo matemático que permite obter uma aproximação de Dii para redes operando com o protocolo passagem de permissão em barra comum com disciplina não exaustiva.

Para a avaliação de Dii no caso da disciplina exaustiva, utilizaremos os resultados obtidos em (BUX83), para aproximação do tempo de retardo em filas com serviço cíclico.

Seja Yi a taxa de chegada em cada estação, hi o tempo médio para trans-

missão de cada pacote, vi o tempo de passagem da permissão entre duas estações consecutivas no ciclo e Pr o número de estações em cada rede.

Então, a sobrecarga em cada ciclo, devido a passagem da permissão entre as estações e ao intervalo de contenção (IC) é dada por:

$$V = IC + \sum_{i=1}^{Pr} vi \qquad (2)$$

A utilização de cada estação para transmissão de mensagens é dada por:

$$pi = Yi\ hi \qquad (3)$$

E a utilização total p é:

$$p = \sum_{i=1}^{Pr} pi \qquad (4)$$

A duração média do ciclo é dada por:

$$C = \frac{V}{1 - p} \qquad (5)$$

O tempo médio de atendimento a uma estação é dado pelo produto do número de mensagens que cheguem em um ciclo (YiC) pelo tempo médio de atendimento.

$$Ti = Yi\ C\ hi \qquad (6)$$

Então, utilizando os resultados obtidos em (BUX83), se obtém o tempo médio de espera em cada estação:

$$Wi = \frac{1 - pi}{2(1-p)} V + \frac{Yi \cdot hi^{(2)}}{2(1-pi)} + \frac{1}{2(1.p)\ (1-pi)} \cdot \sum_{\substack{k=1 \\ k \neq i}}^{Pr} \frac{Yk\ hk^{(2)}\ (1-pi)^{(2)} + Yi\ hi^{(2)}\ pk^2}{1 - pi - pk + 2\ pi\ pk} \qquad (7)$$

Para o caso de carga simétrica (ou balanceada), onde:

$$Yi = \frac{Y}{Pr} \qquad i=1,\ldots,Pr$$

$$hi = h \qquad i=1,\ldots,Pr$$

temos que:

$$Wi = \frac{1 + p/Pr}{2(1-p)} \; V + \frac{Yh^{(2)}}{2(1-p)}$$

Finalmente, o tempo de retardo na rede i, será dado por:

$$Dii = Wi + hi + vi \qquad (8)$$

3 - RESULTADOS NUMÉRICOS

Nesta seção utilizaremos o modelo analítico desenvolvido na seção anterior, para comparar o desempenho do conjunto de redes interligadas, utilizando a disciplina exaustiva, com a disciplina não exaustiva segundo o modelo desenvolvido em (MENA84).

Inicialmente, a figura 2 apresenta as curvas do tempo médio de espera em uma rede isolada versus o número de processadores na rede, com uma taxa de chegada média de mensagens por estação de 10 msg/seg (POISSON), sendo o tamanho médio das mensagens 1600 bits (exponencialmente distribuídos) e uma capacidade de transmissão na barra de 10 Mbits por segundo.

A curva 1 representa os resultados analíticos para a disciplina exaustiva, fazendo o balanceamento das cargas das estações usuárias e do gateway como foi feito para o caso não exaustivo, na obtenção da figura 1.

A curva 2 apresenta os resultados analíticos, considerando a carga real nas estações usuárias e nos gateways e calculando a média do tempo de espera proporcional as taxas de chegada, ou seja:

$$Wmed = \frac{Pr\ Ye\ We + Gr\ Yg\ Wg}{Pr\ Ye + Gr\ Yg} \qquad (9)$$

Onde Pr é o número de estações na rede, Gr é o número de gateways na rede e We e Wg os tempos de espera obtidos através da expressão (7), nas estações e nos gateways, respectivamente.

A curva 3 representa os resultados obtidos através do modelo de simulação, desenvolvido em SIMSCRIPT II e que modela o sistema com poucas simplificações.

A figura 3 apresenta a comparação entre os resultados obtidos, usando a disciplina exaustiva e não exaustiva, para o retardo fim-a-fim no conjunto de redes interligadas, utilizando as topologias em ANEL E ESTRELA, para um total de 320 estações com uma taxa de chegada média de 10 msg/seg (POISSON), sendo o tamanho médio das mensagens 1600 bits (exponencialmente distribuídos) e a capacidade de transmissão na barra de 10 Mbits por segundo, sendo que a probabilidade de comunicação entre dois pares quaisquer de processadores, é a mesma para todos os pares. Esta curva foi obtida com o modelo analítico, a menos dos resultados pra o caso não exaustivo, com duas redes, que foi obtido através de simulação.

A figura 4 apresenta os resultados obtidos para o mesmo caso da figura 3, com a diferença que a probabilidade de comunicação entre dois pares de processadores não é uniforme. Em vez disto, a probabilidade de comunicação entre dois pares é tal que, pares onde os processadores são próximos, tem uma maior probabilidade de comunicação do que os pares formados por processadores distantes.

A análise das figuras 3 e 4 nos indica que a disciplina exaustiva é um pouco melhor que a não exaustiva, pois o retardo no gateway é menor na disciplina exaustiva. Isto se deve ao fato que na disciplina não exaustiva, cada mensagem que atravessa o gateway deve esperar a passagem da permissão entre as demais estações da rede.

Finalmente, a figura 5 apresenta a comparação entre o tempo de espera nas estações usuárias e nos gateways, considerando a carga real, para o mesmo ambiente descrito na obtenção da figura 2. É interessante observar que embora a carga no gateway seja muito maior que a carga nas estações usuárias (Ye= 10 msg/seg e Yg= 400 msg/seg) o tempo médio de espera no gateway é inferior ao tempo médio de espera nas estações usuárias. Isto se deve ao fato que, com a disciplina exaustiva, as mensagens que chegam e encontram uma transmissão corrente no gateway, são transmitidas sem sofrer a sobrecarga de espera da chegada da permissão, sendo que esta diferença é tão maior quanto maior seja o número de estações na rede e quanto mais desbalanceado seja o tráfego.

4 - CONCLUSÕES

Este artigo compara as disciplinas de atendimento exaustiva e não exaustiva na interligação de redes locais que utiliza o protocolo de passagem de permissão. A principal conclusão é que para conjuntos de redes com poucos processadores por rede (até 80 processadores nos casos analisados) não existem diferenças significativas no retardo das mensagens com uma ou outra disciplina. Para a topologia de interligação em estrela, os resultados são praticamente os mesmos. Para a topologia de interligação em anel a diferença nos casos analisados foi de menor que 10%.

Para conjunto de redes com muitos processadores por rede (acima de 80 processadores nos casos analisados) a disciplina exaustiva é mais adequada que a não exaustiva. Isto se deve ao fato que a última introduz um retardo muito grande nas mensagens que atravessam os gateways. Este retardo é causado pela passagem da permissão de acesso entre todas as estações da rede, para cada mensagem que atravessa os gateways.

REFERÊNCIAS BIBLIOGRÁFICAS

(BUX83) BUX, W e Truong, H., "MEAN-DELAY APPROXIMATION FOR CYCLIC - SERVICE QUEUEING SYSTEMS", Performance Evaluation 3, 1983.

(LELL83a) Lellis, L e Menascé, D., "MODELAGEM APROXIMADA DE REDES LOCAIS EM BARRA COMUM", ANAIS X SEMISH, Campinas, Jul. 1983.

(LELL83b) Lellis, L e Menascé, D., "ANÁLISE DE DESEMPENHO DE REDES LOCAIS COM PASSAGEM DE PERMISSÃO", RBC, Vol. 3, Num. 1, 1983/1984.

(LELL84) Lellis, L e Menascé, D., "AVALIANDO O DESEMPENHO DE REDES INTERLIGADAS", ANAIS XI SEMISH, Vicosa, Jul. 1984.

(MENA84) Menascé, D e Lellis, L., "PERFORMANCE EVALUATION OF ISOLATED AND INTERCONNECTED TOKEN BUS LOCAL AREA NETWORKS", PROC. SIGMETRICS, Boston, Set. 1984.

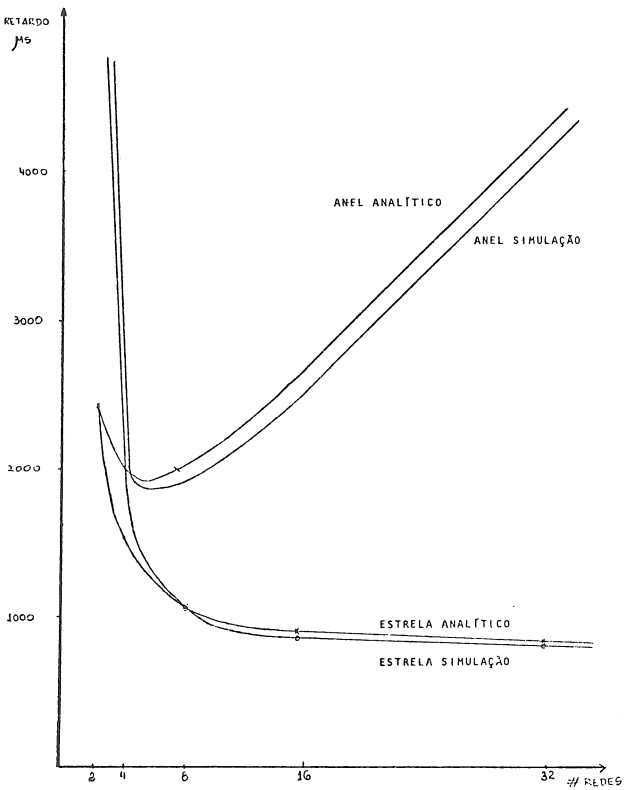


FIGURA 1 – Retardo versus número de Redes Interligadas utilizando a Disciplina não Exaustiva

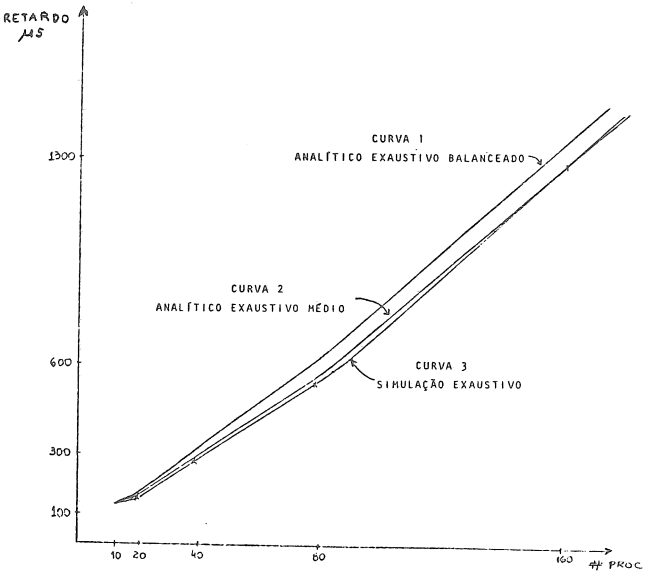


FIGURA 2 - Tempo Médio de espera em uma Rede versus o número de Processadores na rede p/ Disciplina Exaustiva

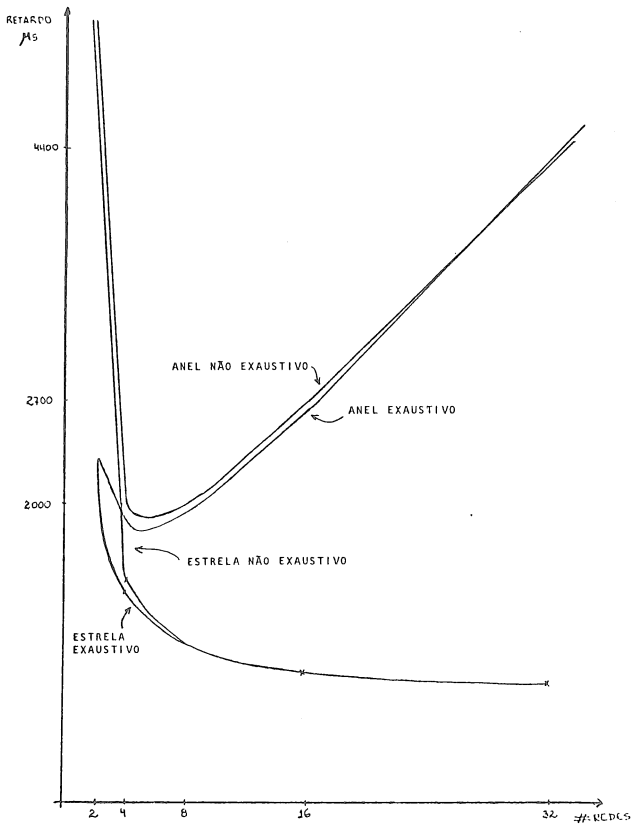


FIGURA 3 - Retardo versus número de Redes Interligadas
Disciplinas Exaustiva e Não Exaustiva

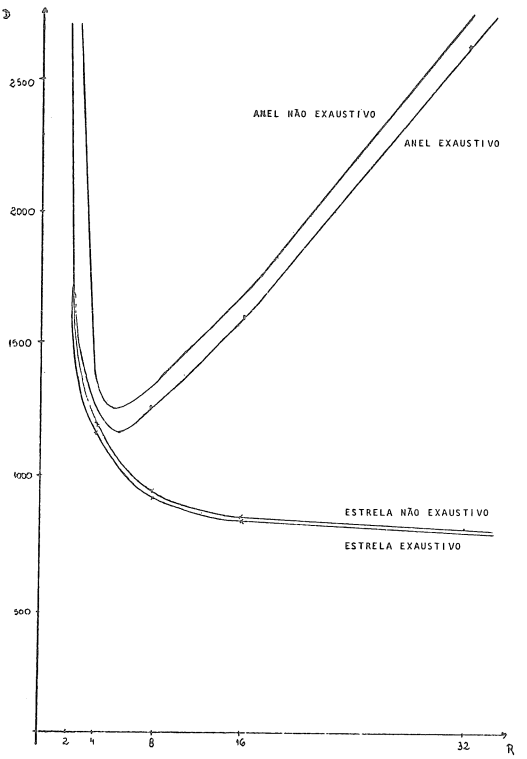


FIGURA 4 - Retardo versus número de Redes Interligadas Disciplinas Exaustiva e Não Exaustiva

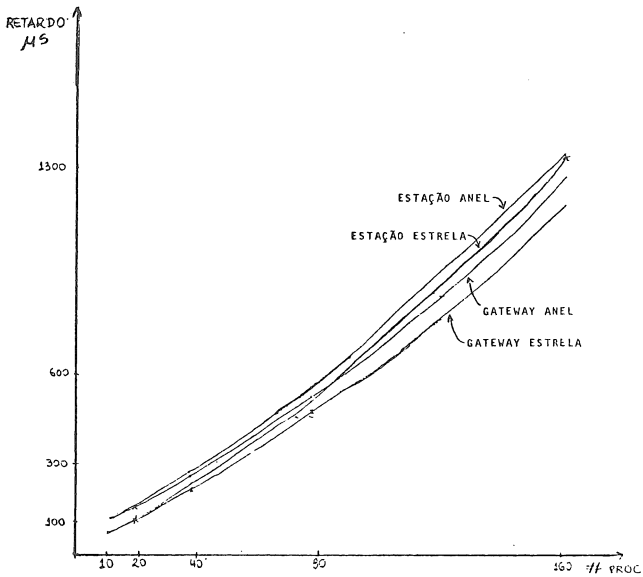


FIGURA 5 - Tempo de Espera nas Estações Usuárais e nos Gateways em função do número de Processadores na Rede

UMA REDE LOCAL EM ANEL PARA O DCC

A.A.F. LOUREIRO*; D.C. SILVA JUNIOR**; J.E.R. DANTAS***; N.A.C. LAGES****

SUMARIO

Este artigo descreve o projeto de uma rede local piloto para a interligação de estações de trabalho e microcomputadores para o desenvolvimento de aplicações científicas no Departamento de Ciência da Computação (DCC). E apresentada uma rede em anel, com controle de acesso por passagem de token a uma taxa de 850 Kbps de acordo com o futuro padrão IEEE 802.4. A seguir, é feita uma descrição das características e requisitos de um Sistema Operacional de Rede (SOR). Finalmente, são feitas considerações sobre uma primeira implementação do projeto.

ABSTRACT

This paper describes the design of an experimental Token Ring Network for the interconnection of workstations and microcomputers at the Department of Computer Science according to IEEE 802.4 proposed standard. The requirements for a Network Operating System are also described as well as the aspects of an initial implementation of the project.

* Bacharel em Ciência da Computação (UFMG, 1983); mestrando em Ciência da Computação; Sistemas Operacionais e Redes de Computadores; Analista de Sistemas do Laboratório de Computação Científica; UFMG/ICEx/LCC - Caixa Postal 702; Belo Horizonte - MG - 30000.

** Engenheiro Eletricista (EEUFMG, 1983); mestrando em Ciência da Computação; Arquitetura de computadores, Redes Locais e Automação; Pesquisador do Depto. de Ciência da Computação; UFMG/ICEx/DCC - Caixa Postal 702; Belo Horizonte - MG - 30000.

*** Engenheiro Mecânico (EEUFMG, 1969); PhD em Ciência da Computação (UCLA, 1981); Professor Adjunto do Depto. de Ciência da Computação; Sistemas Operacionais, Redes de Computadores, Sistemas de Banco de Dados, Modelagem de Sistemas; UFMG/ICEx/DCC - Caixa Postal 702; Belo Horizonte - MG - 30000.

**** Engenheiro Eletricista (EEUFMG, 1973); Doutor em Engenharia Elétrica (UNICAMP, 1981); Professor Adjunto do Depto. de Ciência da Computação; Controle de Processos em Tempo Real, Redes Locais de Computação Distribuida, Arquitetura de Computadores; UFMG/ICEx/DCC; Caixa Postal 702; Belo Horizonte - MG - 30000.

1. Introdução

Com a diminuição de custos e o aumento na complexidade de dispositivos LSI, as mudanças na forma em que a informação é obtida, processada e usada tem sido grande. Este fato pode ser observado no aumento de pequenos sistemas de computação dedicados com computadores pessoais, processadores de texto e terminais inteligentes. Estes pequenos e dispersos sistemas são mais acessíveis e mais fáceis de usar que um computador de grande porte.

Como o número destes pequenos sistemas aumentam em uma organização, surge a tendência a interligação deles devido principalmente:

.Compartilhamento de recursos caros, como impressora de linha e disco tipo 'Winchester'.

.Troca de informações entre sistemas, como o acesso a dados e programas de outros usuários.

Tal interligação tende a ser restrita a uma sala de escritório, um prédio, um instituto de uma universidade ou uma fábrica. Estas redes são conhecidas como Redes Locais. Neste ponto podemos definir uma rede local como :

Uma Rede Local é uma rede de comunicação que provê interconexão de uma variedade de dispositivos de comunicação de dados restrita a uma pequena área /STALLINGS 84/.

Desta definição podemos tirar três elementos importantes:

.Uma rede local é uma rede de comunicação e não uma rede de computadores.

.A frase "dispositivos de comunicação de dados" deve ser interpretada amplamente, de modo a incluir : computadores, terminais, periféricos, sensores (umidade, temperatura, etc.), voz digitalizada, imagens de televisão, fac-símile e gráficos.

.O escopo geográfico da rede local está restrito a uma pequena área.

Uma observação interessante é que a rede local é geralmente uma rede privada e não pública ou comercial.

Algumas das principais características de uma rede local, são :

.Alta taxa de comunicação. (0,1 a 100 Mbps)

.Pequenas distâncias. (0,1 a 10 Km)

.Baixa taxa de erros. (menor que 1 bit em 1 Tera bytes)

As características das redes locais até agora vistas, podem corresponder aos barramentos de computadores também. Segundo Clarck a distinção não é topológica, tecnológica ou geográfica, mas sim filosófica. Um barramento de computador é voltado para a interligação de componentes de um único sistema de computação. É bem difícil a um computador continuar o seu processamento na ausência de seu barramento. Ao contrário, uma rede local pode ser vista como a interligação de vários nodos autônomos, sendo que cada um pode operar por si próprio na ausência da rede /CLARCK 78/.

2. A Rede DCC

A nossa proposta de rede local para o DCC, pode ser definida como uma rede em anel, com controle de acesso ao meio por passagem de token, usando como meio de transmissão o par trançado e sendo controlado por um microprocessador Z80-A. Pode se notar a preocupação em ficar dentro do futuro padrão IEEE 802.

Esta rede tem no momento como principal objetivo a efetivação de uma rede local piloto no âmbito do DCC. Desta forma pode ser considerada como uma rede experimental para teste de algoritmos de processamento distribuido e controle de tempo real. Para não criar mais uma rede diferente de tantas outras que já existem, procuramos como referência o padrão IEEE 802 que por sua vez tem como referência o padrão OSI da ISO /MYERS 82/.

2.1 Topologia

Foi escolhida uma topologia em anel, principalmente pelo fato dela ter um comportamento determinístico, que é fator determinante em aplicações de tempo real. Como vantagem decorrente, é a topologia que mais facilmente se adapta a qualquer meio de transmissão, podendo no futuro utilizar fibra ótica no lugar do par trançado. Embora a rigor, a topologia mais adequada ao controle de processos em tempo real seja o barramento com controle por token, a rede DCC se presta ao desenvolvimento de ferramentas para esta aplicação.

Alguns estudos de desempenho mostram que as redes em anel-token não são muito afetadas com elevadas taxas de chegada de mensagem, tamanho da mensagem ou tamanho do anel /BUX 81/ e /STUCK 83/.

2.2 Controle de Acesso ao Meio

Este nível define um nível de enlace independente do meio, montado sobre um Nível Físico que é dependente do meio. As principais funções deste nível, serão :

.Encapsulamento de Dados
- enquadramento
- endereçamento
- deteção de erros

.Gerenciamento do Enlace
- alocação do canal
- perda e/ou duplicação do token

Para a execução da primeira sub-camada, o Encapsulamento de Dados, usaremos um subconjunto do protocolo HDLC /TANENBAUM 81/ voltado para comunicação balanceada com até 255 nodos, sendo um endereço destinado a mensagem do tipo difusão, e até 256 processos por nodo.

O Gerenciamento do Enlace será feito usando o método de passagem de token. Foi escolhido porque permite um controle do anel de maneira descentralizada e é economicamente mais viável que o método de inserção de registro /LIU 78/. A alocação do canal só é possível quando o nodo estiver de posse do token. Este token é um quadro composto dos campos de Flag , um campo de controle com código de token livre e um campo de FCS. Um nodo ao tomar posse do token terá um tempo máximo de posse dele, permitindo assim determinar o comportamento do anel. Ao transmitir uma mensagem, o nodo originador será responsável por retirar a mensagem do anel, garantindo assim o conhecimento da integridade e a não existência de erros na transmissão. Na ocorrência desta última possibilidade o nodo originador retransmitirá a mensagem. O nodo destino ao reconhecer o seu endereço na mensagem copia e retransmite a mensagem . Após recebê-la completamente adiciona no fim da mensagem um quadro de confirmação de 8 bits indicando a condição de recepção da mensagem. Para enviar este quadro o nodo destino tem um tempo máximo para transmiti-lo. Passado este tempo, o nodo originador reconhecerá a não recepção da mensagem pelo nodo destino e a retransmitirá. Havendo nova falta de campo de reconhecimento, ele considerará o nodo como desligado e notificará os demais nodos. No caso de mensagem tipo difusão, o nodo ao reconhecer o endereço de difusão copiará e retransmitirá a mensagem sem adicionar o campo de confirmação. Ao terminar a transmissão de suas mensagens o nodo passará o token para o nodo seguinte.

No tocante a perda do token, um nodo ao passá-lo para o próximo nodo, inicializa um temporizador com o tempo máximo de utilização da rede pelos outros N-1 nodos, já que ela é determinística. Ao findar este tempo e o nodo não tiver recebido o token, ele reconhecerá a perda dele e irá introduzir um novo token na rede. O ajuste do tempo máximo de utilização garante a deteção da perda do token entre o último nodo adjacente e o próprio.

Com relação a duplicação do token, pela definição da alocação do canal, torna-se uma tarefa muito fácil. Enquanto o nodo está transmitindo, ele é responsável pela retirada da mensagem do anel. Assim, enquanto retira a mensagem o nodo a compara com a transmitida. Caso elas não sejam iguais, houve uma duplicação de tokens e existem duas mensagens circulando no anel. Esta situação leva o anel a um estado de contenção /CLARCK 78/.

Sendo detetada a duplicação do token, entra-se no estado de contenção do anel onde os nodos param de transmitir e continuam a retirar as mensagens do anel até este ficar ocioso. Então ambos os nodos disparam seus temporizadores com tempo de utilização da rede e vão para o estado de espera por token.

2.3 Nível Físico

O Nível Físico provê um canal físico de 850 Kbps em um par trançado. Ele deve ser capaz de mostrar ao nível superior um canal lógico capaz de transmitir e receber bits. As funções principais deste nível, são :

.Codificação dos Dados
- geração/remoção de preâmbulos.
- codificação/decodificação dos bits.

.Acesso ao Canal
- transmissão/recepção dos bits.

A codificação dos bits será feita utilizando o código Manchester II /SANDERS 81/.

O acesso ao canal será feito através do protocolo de nível físico RS 422-A. Este protocolo define linhas eletricamente balanceadas para comprimentos máximos de 1 Km e taxa de até 10 Mbps /STALLINGS 84/. Para assegurar a maior imunidade a interferências externas, será usado um par trançado blindado. Para este acesso serão usados acionadores e receptores de linha para RS 422-A em circuito integrado.

2.4 Enlace Lógico

Este nivel suporta as funções de enlace de dados que são independentes do meio usado. Suas principais funções, são :

.Servir como interface entre o hospedeiro e o nodo da rede.
.Ser o responsável pela inicialização do nodo na rede.
.Organizar o fluxo de dados.
.Controlar o tratamento e recuperação de erros.

3. Implementação do Nodo

Cada nodo será constituído de duas partes : o Módulo do Processador (MP) e o Controlador de Enlace Serial (CES).

O Módulo do Processador como visto na figura 1, utiliza um microprocessador Z80-A, 4 temporizadores de 16 bits i8253, 24 bits de interface paralela programável i8255, memória RAM e EPROM e controlador de interrupções. A interface paralela será usada na ligação com o hospedeiro, usando um protocolo simples implementado em hardware. A interface serial que se vê no MP, só será usada durante a fase de desenvolvimento e depuração do projeto. Na EPROM será colocado programa de controle do nodo e a RAM será usada para o armazenamento de mensagens de/para o hospedeiro. Os temporizadores serão usados para a retransmissão de mensagens não reconhecidas e gerenciamento do token.

O Controlador de Enlace Serial é composto pelo MPSC, um controlador de DMA i8237, memória RAM e o controlador de acesso ao canal. O MPSC /ZILOG 78/ é o responsável pelo protocolo HDLC, com deteção de endereço específico ou de difusão, bit 'stuffing', cálculo e verificação de CRC (do campo FCS), pelo envio e retirada de preâmbulos e pela transmissão e recepção de bits. A taxa de comunicação de 850 Kbps é imposta pela limitação desta MPSC. O CES tem um barramento separado do barramento do MP, de modo que ambas partes podem operar sem interferir no processamento da outra. Para implementar esta possibilidade, existirá uma porta que controlará o intercâmbio entre os dois barramentos. Quando o nodo está transmitindo, o DMA tem que retirar caracteres da fila da RAM do CES e colocar no MPSC e retirar caracteres que chegam ao MPSC e enviá-los ao MP para comparação. O controle e acesso ao canal é responsável pela codificação e decodificação do sinal binário do MPSC em sinal Manchester II e pela transmissão e recepção deste sinal binário no protocolo RS 422-A. O controle de acesso ao canal dispõe ainda de um relé que é fechado quando o nodo está desligado e aberto quando ele está ativo, de

modo que o anel continue ativo e dispense uma interface ativa com repetidor em cada nodo.

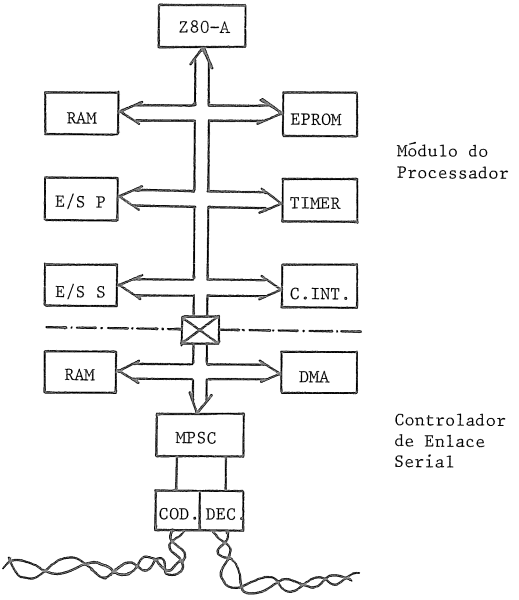


Figura 1

4. O Sistema Operacional da Rede

4.1 Introdução

Uma rede local consistindo de microcomputadores permitirá que recursos sejam compartilhados, sejam eles informação, poder computacional ou periférico de custo elevado. A operação eficiente desses recursos requer um projeto cuidadoso da interface entre o programa do usuário e a rede. O Sistema Operacional da Rede (SOR), neste caso, é o elemento fundamental que permite que os usuários tenham acesso e uso dos recursos da rede. O SOR pode ser visto como o agente colocado entre o usuário e a rede que deve prover, em termos gerais:

.acesso fácil e uniforme aos recursos da rede (computacionais e informação);

.ser o mais independente possível do sistema operacional do hospedeiro;

permitir que o usuário tenha facilidade em definir e usar novos serviços ("user-friendly").

O projeto de um SOR torna-se mais complicado se os computadores na rede têm diferentes caracteristicas de arquitetura. Muitos SOR começaram a ser investigados desde a última década /THOMAS 73/, /COSELL 75/, /RETZ 76/, /ROBINSON 77/, /FORSDICK 78/, /LIU 78/.

Na maioria das redes existentes, os protocolos de alto nível são voltados para funções especificas, tais como protocolos de terminal virtual e transferência de arquivos. Estes protocolos utilizam os serviços do protocolo de comunicação entre processos. Todo o potencial oferecido por uma rede de computadores para compartilhamento de recursos e processamento distribuido não é utilizado devido as caracteristicas de arquitetura dessas redes que, em geral, têm os seguintes pontos em comum:

.nenhum suporte básico para criação de novos recursos ou serviços fora dos existentes;

.um usuário ao desejar criar ou usar um novo recurso deve preocupar-se com toda a estrutura (conversão de dados, comando, proteção, controle de erro, gerenciamento, sincronização, interface com o protocolo de comunicação entre processos, etc.) que será necessária na sua utilização;

.o usuário deve conhecer os diferentes nomes e outros mecanismos de acesso para cada computador (host), serviço, etc.

Somente nestes últimos anos é que alguns desses problemas têm sido resolvidos, com proposições de novas arquiteturas /DONNELLEY 79/, /WATSON 80/, /CHESLEY 81/.

4.2 Caracteristicas de um SOR

Para permitir que haja um compartilhamento uniforme dos recursos e informação na rede, é necessário que cada sistema execute uma estrutura comum chamada SOR. Cada sistema é responsável pela conversão entre uma representação local e de rede para todas as fontes de informação, ou seja, tanto do usuário como do sistema. Isto faz com que o usuário tenha a ilusão de ter um sistema poderoso com muitos recursos disponiveis. Dessa forma o usuário não precisa ter que saber a localização dos recursos, a arquitetura e o método de operação.

Alguns requisitos e filosofias de projetos podem ser enumerados para um SOR :

.O SOR deve suportar uma variedade de sistemas operacionais,

tal que a sua adição não cause modificações extensivas no sistema operacional (SO) do hospedeiro. Isto pode ser conseguido se o implementarmos como um componente separado que é visto como uma interface entre os processos dos diversos SO que estão comunicando e cooperando para alcançar os objetivos desejados.

.As requisições devem ser tratadas localmente, quando possíveis. Caso contrário, elas devem ser convertidas para uma mensagem de formato padrão e enviadas para a rede para algum componente remoto que ficará encarregado de executar o serviço e retornar o resultado. Vendo do ponto de vista do outro sistema, ele deve ser capaz de aceitar ou rejeitar um serviço. Se uma requisição é aceita por vários componentes remotos, a estação que emitiu o pedido pode escolher uma política qualquer de quem irá executá-lo.

.O SOR deve suportar transparência de localização. Isto significa que um processo pode acessar qualquer tipo de recurso sem ter que saber a sua localização.

.O SOR deve prover uma gerência de processos de tal forma que um par de processos possam se comunicar.

.Deve haver uma representação padrão já que a representação interna dos dados são, em geral, diferentes em cada hospedeiro.

.O SOR deve ser modular o suficiente para permitir que avanços tecnológicos possam ser incorporados.

4.3 Ambiente de Operação da Rede

O Departamento de Ciência da Computação (DCC) está desenvolvendo Estações de Trabalho para o Desenvolvimento de Aplicações Científicas, com as seguintes características: UCP i8086, memória de 512 kbytes, interfaces seriais e paralelas, capacidade para múltiplos mestres no barramento Multibus, terminal gráfico de média/alta resolução, sistema operacional DCC/IX (compatível com o UNIX versão 7). Estas estações serão reproduzidas e distribuídas para os pesquisadores do DCC, para o desenvolvimento de projetos de ensino, pesquisa e extensão.

O objetivo básico deste projeto é desenvolver uma rede local interligando as estações de trabalho e os computadores do DCC : Nexus-1600, Quartzil QI-800 e Poly 105.

Numa segunda fase, pretende-se interligar esta rede a um IBM 4341.

5. Propostas e perspectivas do Projeto

Quanto a implementação do nodo, o que se pretende é obter uma rede local piloto no DCC. Pode ser constatada uma grande perda quanto ao desempenho dela, se comparada com redes em anel encontradas na literatura. As limitações foram expostas, e qualquer melhora na implementação, irá requerer um microprocessador mais rapido com um conjunto de instruções mais poderoso, ou um microprocessador em bit slice. Porém, a sua implementação fornecerá uma apreciável bagagem de conhecimentos para um futuro projeto mais elaborado.

Não se pretende inicialmente, que o SOR a ser projetado tenha todas as caracteristicas que foram enumeradas no item 4.2.

Nesta primeira fase pretende-se :

.Desenvolver um software para a conexão da sub-rede de comunicação : protocolo de transporte e interface com o SO.

.Implementação de um protocolo para transferência de arquivos entre os diversos modelos de computadores ligados a rede.

Esta rede, ainda na primeira fase, irá dar suporte para o estudo de controle de processos distribuidos. Numa etapa posterior, pretende-se utilizar o IBM 4341, já citado, como servidor de arquivo para a rede.

6. Bibliografia

/BUX 83/ - Bux, W., "Local Area Subnetworks : A Performance Comparison," IEEE Trans. on Comm., vol 29, no 10, Oct. 1981, pp. 1465-1473.

/CHESLEY 81/ - Chesley, H.R. & Hunt,.B, "Squire - A Communications-Oriented Operating System", Computer Networks, vol 5, 1981, pp. 333-339.

/CLARCK 78/ - Clarck, D.D. et.al, "An Introduction to Local Area Networks," Proc. of IEEE, vol 66, no 11, Nov 1978, pp. 1497-1517.

/COSELL 75/ - Cosell, B.P. et al, "An Operating System for Computer Resource Sharing", ACM Oper. Syst. Rev., vol 9, no 5, 1975, pp. 75-81.

/DONNELEY 79/ - Donnelley, J., "Components of a Network Operating System", Computer Networks, vol 3, 1979, pp. 389-399.

/FORSDICK 78/ - Forsdick, H.C. et al ,"Operating System for Computer Networks," Computer, vol 11, no 1, 1978, pp. 48-57.

/LAMPSON 81/ - Lampson, B.W., "Distributed Systems-Architecture and Implemantation", Springer Verlag, Berlin, 1981.

/LIU 78/ - Liu, M.T., "Distributed Loop Computer Network", Advances in Computers, vol 17, Academic Press, NY, 1978, pp. 163-221.

/MYERS 82/ - Myers, W., "Toward a Local Network Standard", IEEE Micro, vol 2, no 4, Aug 1982, pp. 28-45.

/RETZ 76/ - Retz, D.L. & Schafer, B.W., "Structure of the ELF Operating System", AFIPS Natl. Comp. Conf. Expo., Conf. Proc., 1976, pp. 1007-1016.

/ROBINSON 77/ - Robinson, R.A., "National Software Works: Overview and Status", Proc. COMPCOM Fall 1977, pp. 270-273.

/SANDERS 81/ - Sanders,.L., "Manchester II Transfers Data with Integrity, Speed", Electronic Design, March 19, 1981, pp. 233-238.

/STALLINGS 84/ - Stallings, W., "Local Networks", Computing Surveys, vol 16, no 1, March 84, pp. 3-41.

/STUCK 83/ - Stuck, B.W., "Calculating the Maximum Mean Data Rate in Local Area Networks", Computer, vol 16, no 5, May 1983, pp. 72-76.

/TANENBAUM 81/ - Tanenbaum, A.S., "Computer Networks", Prentice Hall Inc., NJ, 1981, pp. 167-172.

/THOMAS 73/ - Thomas, R.H., "A Resource Sharing Executive for the ARPANET", AFIPS Natl. Comp. Conf. Expo., Conf. Proc., 1973, pp. 155-163.

/WATSON 80/ - Watson, R.W. & Fletcher,.J.G., "An Architecture for Support of Network Operating System Services", Computer Networks, vol 4, 1980, pp 33-49.

/ZILOG 78/ - "Z80 SIO Technical Manual", Zilog Inc., CA,1978.

L.F.FRIEDRICH*, S.S. TOSCANI**, M.TORNQUIST***

SUMARIO

O artigo descreve o software de comunicação para uma rede local de microcomputadores que está sendo construida na UFRGS. A rede é do tipo barramento (ethernet) com controle de acesso distribuido. O software está operacional, em fase de testes.

ABSTRACT

The paper describes the communitation software for a microcomputer local network that is being built at UFRGS. The network is ethernet like with distributed access control. The software is under testing.

*Engenheiro Mecânico (UPF,1978), aluno do Curso de Pós-Graduação em Ciência da Computação da UFRGS, concluindo sua dissertação de mestrado. Áreas de interesse: sistemas operacionais distribuidos e redes de computadores. CPGCC/UFRGS - Caixa Postal 1501 - Porto Alegre - RS - CEP 90.000.

**Engenheiro Eletricista (UFRGS,1967), Mestre em Informática (PUC/RJ,1969) Áreas de interesse: programação concorrente e sistemas operacionais. Professor Adjunto do DI/CPGCC da UFRGS. Caixa Postal 1501 - Porto Alegre-RS CEP 90.000.

***Tecnólogo em Processamento de Dados (UFRGS,1977), Mestre em Ciência da Computação (UFRGS,1982). Áreas de interesse: MUMPS, sistemas distribuidos e validação de sistemas. Professor assistente do DI/CPGCC da UFRGS. Caixa Postal 1501 - Porto Alegre - RS - CEP 90.000.

1. INTRODUÇÃO

Este artigo apresenta o software de comunicação para uma rede local de microcomputadores que está sendo construida no Curso de Pós-Graduação em Ciência da Computação de UFRGS.

Os microcomputadores utilizados são os micros MAXXI da Polymax e a rede local é do tipo barramento (ethernet) com controle de acesso distribuido. O hardware de comunicação consiste basicamente de uma interface serial RS-232-C, com a inclusão de um pequeno circuito no conector DB-25, conforme descrito em |CLE 81|. O software foi escrito em linguagem de montagem Z-80, no sistema CP/M, ocupando cerca de 5 Kbytes de memória, e pode ser utilizado a partir de qualquer linguagem de alto nível que permita chamadas de subrotinas escritas em Assembler.

O software deverá ser embutido, mais tarde, no interpretador MUMPS/M, já implementado nos micros MAXXI |TOR 83|, originando uma rede local MUMPS. Esta rede colocará à disposição dos usuários todas as facilidades para o desenvolvimento de aplicações normalmente oferecidas pelos sistemas MUMPS.

O software baseia-se nas recomendações da ISO para interconexão de sistemas abertos (OSI) |DAT 81| e foi influenciado pelo trabalho desenvolvido no projeto da rede local do IME |FER 84|. O sistema segue a filosofia de divisão em níveis, sendo composto por dois níveis apenas: Nível de Enlace (NE) e Nível de Transporte (NT).

O Nível de Enlace implementa as funções necessárias à transmissão/recepção das mensagens de enlace (quadros) e oferece estes serviços através de primitivas que são acionadas pelo Nível de Transporte.

O Nível de transporte tem como função oferecer ao Nível Superior (NS) os serviços necessários para o estabelecimento de circuitos virtuais entre os micros comunicantes. O Nível de Transporte assegura o transporte das mensagens do Nível Superior de forma ordenada e segura.

No restante deste trabalho o software de comunicação é apresentado de forma detalhada.

2. NÍVEL DE ENLACE

O nível de enlace (NE) tem como responsabilidade principal o controle da transferência das mensagens enviadas pelo nível imediatamente superior. Para que o NE possa enviar mensagens para a rede é necessário que exista uma política de acesso ao meio de comunicação. O método que foi usado é o CSMA/CD ("Carrier Sense Multiple Acess/Colision Detect") |MET76|.

Neste método a portadora inicialmente é sentida e, estando desocupada, o NE inicia a transmissão. A seguir é verificada a existência de colisão (há colisão se o que uma estação está enviando está sendo recebido por ela mesma de forma não consistente). No caso em que há colisão a transmissão é interrompida e uma nova tentativa de transmissão é realizada.

O NE transmite os dados byte a byte para uma interface serial de comunicação com padrão RS-232C. A interação do NE com o hardware é feita através da monitoração dos sinais que são fornecidos por essa interface. Estes sinais são registrados na posição de memória correspondente ao byte de estado da interface de comunicação.

Uma outra posição de memória armazena os dados que são enviados para a interface, ou retirados da mesma.

No byte de estado são mantidas as seguintes informações necessárias ao NE:

RDRF - "Receive Data Register Full"
Indica o estado da recepção:
Ø - dado recebido já foi lido, ou seja, não chegou dado novo;
1 - dado recebido ainda não foi lido.

TDRE - "Transmit Data Register Empty"
Indica o estado de transmissão:
Ø - dado ainda não foi transmitido, nova transmissão não habilitada;
1 - dado já foi transmitido, habilitada nova transmissão.

DCD - "Data Carrier Detect"
Indica o estado da linha de transmissão:
Ø - linha de transmissão ocupada;
1 - linha de transmissão livre.

Sendo este nível o responsável pela interação com o hardware, o seu desenvolvimento realizou-se de forma simples, procurando facilitar, assim, a execução de alterações quando da modificação do hardware.

O Nível de Enlace oferece seus serviços para o nível imediatamente superior através de duas primitivas (TRANSMISSÃO e RECEPÇÃO) que serão descritos adiante.

2.1. Estrutura de dados

Todas as mensagens que fluem no sistema implementado tem passagem pelo NE, o qual realiza a operação desejada (Transmissão ou Recepção). Pa-

ra isto é utilizada uma estrutura de dados bem definida onde as mensagens são armazenadas.

A área de dados existente no NE serve de buffer tanto na recepção quanto na transmissão dos dados. Esta área do NE é composta por duas partes, a saber: o cabeçalho da mensagem e o texto (dados) da mensagem.

O cabeçalho da mensagem é composto por no máximo seis campos, dos quais quatro apenas são vistos pelo NE; o restante do cabeçalho é de interesse do nível de transporte.

Os campos do cabeçalho de interesse do NE são:

Campo	Tamanho	Conteúdo
Destino	1 byte	endereço destino da mensagem (0 - 255)
Origem	1 byte	endereço origem da mensagem (0 - 254)
Bytestx	2 bytes	número de bytes que compõe a mensagem (cabeçalho do nível superior mais dados)

O endereço (nº de estação) 255 é reservado para indicar mensagem de difusão (todas as estações podem receber a mensagem).

O endereço de origem é obtido na inicialização do sistema, quando informa-se o nº da estação de trabalho.

O campo Bytestx, como indicado, controla a transmissão e/ou recepção do NE a partir do quinto campo do cabeçalho, onde estão informações que dizem respeito ao nível de transporte.

Todas as mensagens do sitema, independentemente do tipo (dados,controle), possuem um cabeçalho fixo que sempre é transmitido. Desta forma o NE transmite/recebe 4 bytes mais os bytes indicados no campo Bytestx.

Os Dados da Mensagem compõe uma parte variável da mensagem, podendo ter no mínimo Ø e no máximo 255 bytes.

Este tamanho foi escolhido em função das características da rede implementada, e pode ser mudado se necessário.

O NE tem ainda a função de anexar a cada mensagem a ele enviada o caracter de início de transmissão "SOH" (start of header).

2.2. Tratamento de Colisões

Na ocorrência de colisão quando da tentativa de transmissão de men

sagens, as estações envolvidas entram em um período de espera randônico, implementado através de um busy wait. Este tempo randônico é conseguido através do registrador R ("REFRESH") da maquina Z-80.

O NE decrementa o valor do registrador R até chegar a zero e, então, volta à tentativa de iniciar uma transmissão. À medida que as colisões vão ocorrendo o tempo de espera aumenta, porque passa a ser o número de colisões ocorridas multiplicado pelo valor do registrador R.

O tratamento de colisão preve um número máximo de colisões que podem ocorrer durante a transmissão de uma mensagem do NE.

O registrador R tem o seu valor modificado constantemente, sendo isto uma característica do microprocessador Z-80.

2.3. Primitivas

O NE fornece serviços ao seu nível imediatamente superior (NT) através de duas primitivas. Estas primitivas são controladas pelo próprio NE, o qual é chamado pelo NT, que fornece como parâmetro de entrada o código da primitiva que deve ser acionada. As primitivas do NE foram projetadas para prover os serviços básicos que são necessários para o nível de transporte poder executar a sua tarefa, sendo bastante simples.

2.3.1. Transmissão

A primitiva TRANSMISSÃO tem como função controlar o processo de transmissão de uma mensagem do nível de enlace e retornar o resultado de sua execução.

Inicialmente a primitiva faz a verificação do estado da linha de transmissão através da leitura do estado do interface serial de comunicação. Enquanto a linha estiver no estado ocupado e o tempo que a primitiva tiver para tentar a transmissão não se esgotar, irá permanecer verificando o estado da linha. Quando esgotar-se o tempo e a primitiva não conseguir a transmissão, voltará ao seu chamador indicando como código de retorno o valor 'FF'H (transmissão não realizada).

Quando a linha estiver no estado livre, a primitiva iniciará a transmissão do caracter de início de transmissão e dos três primeiros campos do cabeçalho da mensagem. Nesta etapa serão detectadas as colisões e tratadas da forma descrita anteriormente. Se a primitiva não conseguir executar de forma correta neste estágio, o código de retorno da primitiva será 'FF'H.

Após ter transmitido a parte referente ao cabeçalho da mensagem,a

primitiva TRANSMISSÃO entrará na sua última etapa que é a transferência do texto da mensagem. O número de bytes que devem ser transferidos é indicado no campo "Bytestx". Durante esta etapa qualquer impossibilidade de transmissão que ocorrer implicará no término da execução da primitiva e no retorno do código de erro 'FF'H.

A primitiva terá sucesso quando todos os bytes da mensagem forem transmitidos de forma correta. O código de retorno para a execução correta é o valor '06'H (transmissão realizada).

2.3.2. Recepção

O objetivo desta primitiva é o de controlar a recepção de uma mensagem. Ela recebe como parâmetro de entrada o endereço da estação origem da mensagem a ser recebida e rejeita mensagens originadas por outras estações. A chamada da primitiva com parâmetro de entrada igual a 'FF'H, indica que a mensagem a ser recebida pode ter origem em qualquer estação.

Em uma primeira etapa, a primitiva RECEPÇÃO verifica o estado da linha de transmissão. Se estiver livre, tenta a recepção do primeiro byte de uma mensagem. Esta tentativa se repetirá enquanto não receber nada e o tempo de recepção não se esgotar. O esgotamento do tempo sem que a primitiva tenha concretizado o recebimento ocasiona o término da execução e a devolução de um código de erro ('FF'H).

Na recepção do primeiro byte a primitiva verifica se o mesmo indica o inicio de uma mensagem ("SOH"). Não indicando, significa que o NE está interceptando uma mensagem pela metade. A primitiva então volta a tentar a recepção do indicador de início de uma mensagem.

Quando ocorre o recebimento de byte "SOH", a primitiva tenta a recepção do cabeçalho da mensagem, que é composto pelos próximos 4 bytes.

Do cabeçalho é extraída a informação referente ao tamanho da mensagem a receber (campo "Bytestx"), o qual controla, a partir de então, a execução da primitiva. Todos o bytes recebidos são colocados na área de dados do NE. No final da recepção é verificado se a mensagem recebida é endereçada à estação receptora, comparando o parâmetro da primitiva RECEPÇÃO e o campo "origem" da mensagem. A execução correta desta primitiva retorna ao chamador o código '06'H, significando sucesso na recepção.

3. NÍVEL DE TRANSPORTE

A interface entre o nível superior (aplicação) e o sistema de comunicação é responsabilidade das primitivas do NT. Estas primitivas imple-

mentam os serviços que são oferecidos para o desenvolvimento de aplicações na rede.

3.1. Estabelecimento da Conexão

Nesta fase da comunicação as entidades envolvidas criam um canal virtual entre si, e estabelecem parâmetros para as fases posteriores da comunicação. O estabelecimento de conexão se dá de duas maneiras: o nível superior requisita ao NT uma conexão, ou a estação escutando a linha recebe um pedido de conexão.

3.1.1. Primitiva CONECTA

Esta primitiva é requisitada pelo nível superior quando for necessário estabelecer um circuito virtual para troca de informações com outra estação remota. Para isto acontecer é necessário que a estação remota envolvida esteja em estado de ESCUTA na rede. Esta condição é conseguida quando o nível superior (na estação remota) ativa a primitiva PERGUNTA.

As conexões se realizam na relação uma para uma, ou seja, em um dado momento cada estação pode estar conectada com apenas uma outra estação remota. Desta forma para que o NT acione a primitiva CONECTA, é necessário que a estação envolvida esteja no estado LIVRE.

Como parâmetro de entrada a primitiva recebe o endereço da estação com a qual deve ser estabelecida a conexão. No retorno é desenvolvido ao nível superior o resultado da execução da primitiva CONECTA, o qual pode indicar:

- sucesso: conexão estabelecida;
- insucesso: conexão não estabelecida;
- estado inicial da estação não é LIVRE.

A ocorrência de um insucesso poderá se dar quando a estação remota não aceitar uma conexão, quando não estiver ativa ou quando a estação origem não conseguir enviar o pedido de conexão.

3.1.2. Primitiva PERGUNTA

Para que uma estação da rede local possa receber mensagens, é necessário (por falta de mecanismo de interrupção, no hardware) que a mesma esteja em estado de espera de mensagem. Isto corresponde a escutar a linha de comunicação a espera de pacotes. A primitiva PERGUNTA é ativada pelo nível superior para tentar a recepção de um pedido de conexão ou de uma mensagem de difusão (broadcast).

A execução da primitiva produz um resultado para o nível superior que pode ser:

- nenhuma mensagem foi recebida;
- mensagem de difusão recebida;
- mensagem pedido de conexão recebida e confirmada.

No segundo e no terceiro caso, a primitiva devolve como parâmetro de saída o endereço da estação que enviou a mensagem. No caso de mensagem de difusão são informados ao nivel superior, a posição do buffer onde está a mensagem e o tamanho da mesma.

Na mensagem de pedido de conexão, a primitiva informa ao nivel superior que o pedido foi recebido e confirmado, ou seja, a conexão foi estabelecida com a estação remota.

3.2. Transferência de Dados

Após o estabelecimento de conexão, as estações envolvidas estão aptas a trocarem informações . Nesta fase de transferência de dados, as estações enviam e recebem mensagens.

3.2.1. Primitiva ENVIA

A primitiva ENVIA é ativada pelo NT quando o nível superior deseja a transmissão de informações para a estação remota com a qual se conectou. O endereço da estação que deve receber a mensagem já foi informado quando o nível superior realizou o pedido de conexão. A estação que requisitou a primitiva CONECTA, tem prioridade para enviar a mensagem inicial para a estação remota.

Toda mensagem que é enviada deve receber uma confirmação de recebimento (acknowledgment). Só após a confirmação do recebimento é que uma próxima mensagem pode ser enviada (ou seja, a janela de transmissão de mensagens é igual a 1).

Na chamada da primitiva o nível superior passa como parâmetros o endereço do buffer da mensagem e o tamanho da mensagem.

Recebe no retorno o resultado de execução da primitiva:

- Sucesso: mensagem transmitida e confirmada;
- insucesso: não recepção de confirmação ou problemas na transmissão;
- sem conexão: estação não está conectada.

3.2.2. Primitiva RECEBE

Para uma estação, que tenha estabelecido conexão com outra remota, poder receber dados enviados pela estação remota é necessário que o nível superior ative o NT, chamando a Primitiva RECEBE.

As mensagens de dados seguem uma seqüência rígida, que é verificada na execução desta primitiva. Para cada mensagem recebida corretamente é enviada uma mensagem de confirmação de recebimento, liberando assim a estação remota da necessidade de repetir a mensagem.

Como parâmetro de entrada, o nível superior, na chamada da primitiva, informa o endereço do buffer onde a mensagem deve ser colocada. Como parâmetro de saída a primitiva devolve o código do resultado de execução, o qual pode ser:

- mensagem recebida;
- mensagem não recebida;
- desconexão pedida e realizada;
- estação não está conectada.

Quando a primitiva recebe alguma mensagem de dados, ela devolve para o nível superior o tamanho da mensagem que foi recebida.

3.3. Término da Conexão

Após a fase de transferência de dados entre as estações envolvidas em uma conexão, as mesmas devem se liberar para que novas conexões possam ser estabelecidas. Para isto o serviço de transporte oferece a primitiva DESCONECTA.

3.3.1. Primitiva DESCONECTA

O objetivo da primitiva DESCONECTA é fazer com que as estações conectadas entre si sejam ordenadamente liberadas para novas conexões.

A estação que recebe o pedido de desconexão deve estar em um estado de recepção, para poder aceitar e confirmar a desconexão.

Como parâmetro de resposta a primitiva devolve para o nível superior o resultado da execução, o qual pode ser:

- sucesso: a conexão com a estação remota foi encerrada;
- insucesso: não foi possível encerrar a conexão (falta de recebimento de confirmação, ou impossibilidade de transmissão);
- a estação não está conectada.

3.4. Difusão de Mensagens

Para possibilitar o envio de mensagens para todas as estações que estejam ativas e em estado de escuta na rede, o NT oferece o serviço de difusão.

A primitiva DIFUNDE transmite uma mensagem para todos receptores da rede. Este tipo de mensagem não necessita da fase de estabelecimento de conexão e tampouco espera a recepção de mensagem de confirmação de recebimento.

Como parâmetros de entrada a primitiva recebe o endereço do buffer onde está depositada a mensagem a ser difundida e o tamanho da mesma. O parâmetro de saída informa ao nível superior o resultado da execução da primitiva DIFUNDE:

- sucesso: mensagem de difusão transmitida;
- insucesso: transmissão negativa.

4. CONCLUSÃO

O sistema aqui descrito está sendo testado e tem apresentado bons resultados. A inexistência de mecanismo de interrupção na interface de comunicação impede que se possa obter um melhor desempenho. Pretende-se, em uma próxima etapa, desenvolver um hardware mais adequado para a gerência da comunicação, no qual se tenha um microprocessador próprio na interface da rede, o que permitirá localizar o software aqui descrito nessa interface, liberando o micro principal de todo o gerenciamento do acesso à rede local.

BIBLIOGRAFIA

/CLE 81/ CLEMENTS, Ken and DAUGHERTY, Dane. Ultra-low-cost network for personal computers. Byte, 6 (10), october 1981.

/DAT 81/ DATA processing-open systems interconnection-basic reference model. Computer Networks, 5 (2): 81-118, April 1981.

/FER 84/ FERNANDES, Lucio. Protocolo de transporte para redes locais. Rio de Janeiro, IME, Fevereiro 1984. (dissertação de mestrado).

/MET 76/ METCALFE, R.M. and BOGGS, D.R. Ethernet: distributed packet switching for local computer networks. Communications of the ACM, 19 (7): 395-404, July 1976.

/TOR 83/ TORNQUIST,M., TOSCANI,S.S. e FRIEDRICH, L.F. MUMPS/NET: uma rede local MUMPS para automação de escritórios. IN: CONGRES

SO NACIONAL DE INFORMATICA, 17., Rio de Janeiro, Outubro 1984. Anais. Rio de Janeiro, SUCESU, 1984.

ANNETTE: UNA RED LOCAL ECONOMICA DE ALTO RENDIMIENTO PARA MICROCOMPUTADORES.

P. O'Callaghan (*) y G. Broner (**)

RESUMEN

Se ha diseñado y puesto a prueba, en forma experimental, una red local de microcomputadores, que resulta económica, en cuanto a costo de interfaces físicas, en comparación con las existentes actualmente en el mercado. La red alcanza buena velocidad, pero su manejo no sobrecarga mayormente la unidad central de procesamiento de cada computador. Así los usuarios pueden utilizar sus programas de aplicación sin verse afectados porque paralelamente estan siendo ejecutadas rutinas de envío y recepción de mensajes. Estos logros se deben principalmente a que se utilizan como manejadores de red componentes del computador que disponen de mucho tiempo ocioso, como lo son el controlador de discos y el manejador de acceso directo a memoria.

ABSTRACT

We have designed and tested a new experimental Local Area Network for microcomputers using cheaper interfaces than others currently on the market. The network can operate at reasonable speed without loading the CPUs of the communicating nodes, thus having minimal impact on the perceived performance of user programs. These characteristics are obtained mainly because the network drivers are common systems components which are normally idle most of the time, such as the disk controller and DMA controller.

(*) PhD en Ingeniería de la Computación, Heriot-Watt University, Edimburgo, Escocia. Profesor Universidad Simón Bolívar, Caracas, Venezuela. Director de Expert Computación, Caracas, Venezuela.

(**) Ingeniero Electrónico, Universidad Simón Bolívar, Caracas, Venezuela. Director de Logiciel S.R.L., Apartado 50939, Caracas 1070A, Venezuela.

La mayoría de las redes locales de computadoras son de tipo "broadcast" [1,2,3,4]. En este tipo de red existe un canal de comunicación único compartido por todas las máquinas que comunican, y cada máquina precisa sólo una interfaz física para conectarse a la red. Para transmitir un mensaje éste se envía al canal según un determinado método de acceso, y el mensaje es escuchado por todos, pero sólo es recibido por su destinatario. No existen elementos de conmutación, por lo que este tipo de red resulta más económico que una red tipo "punto a punto" [5].

Las redes locales que actualmente existen en el mercado, utilizan interfaces costosas para conectar cada computador. La razón de su costo reside en la relativa "inteligencia" de dichas interfaces o controladores. Sus funciones son:

Transmisión:
- Aceptar un mensaje a enviar del computador
- Almacenarlo en su memoria interna
- Formatear el mensaje en pequeños trozos o "paquetes" (utilizando un formato preestablecido)
- Agregarle códigos de comprobación de errores
- Serializar los datos y enviarlos por la red

Recepción:
- Detectar si un mensaje está destinado a su máquina
- Recibirlo
- Comprobar que no ha habido ningún error
- Ensamblar el mensaje
- Pasarlo al computador

Un enfoque para obtener redes de bajo costo ha sido utilizar la unidad central de procesamiento (CPU) de cada computador como manejador de la red [10]. Se logra pues tener una interfaz económica, pero el hecho de ocupar al procesador central en la transmisión y recepción de datos a la red presenta las siguientes desventajas:

1) El CPU realiza el trabajo de tomar los datos a enviar, serializarlos para mandarlos a la red, y calcular, para agregar al mensaje, códigos de comprobación de errores. Esto hace que en la transmisión o recepción de datos, el CPU no pueda realizar otras tareas, lo que puede tener bastante impacto en la velo-

cidad del computador percibida por el usuario.

2) En la recepción, además, el CPU es el encargado de "escuchar" por si llega algún mensaje. Esto obliga al CPU a interrumpir sus tareas para escuchar por ciertos intervalos de tiempo, lo que causa dos efectos negativos:

a) El CPU pierde tiempo de procesamiento, aún cuando no haya presencia de mensajes que lo involucren.

b) Como el CPU sólo será capaz de escuchar por cierto tiempo, para poder realizar sus otras tareas, habrá frecuentes intentos de transmisiones que no se reciben, lo que da lugar a un alto tiempo de establecimiento de la comunicación.

Varios de los efectos negativos del modelo anterior pueden ser solucionados asignando más inteligencia a la interfaz, y por tanto menos funciones al CPU. De esta forma se puede lograr una gran gama de posibilidades en las cuales se descarga de más o menos funciones al procesador central. Por supuesto que al aumentar el número de funciones que realiza la interfaz, se está elevando el costo de la misma, por lo que se enfrenta una relación de compromiso "ineludible".

En el diseño que presentamos a continuación, se logra eludir el mencionado compromiso, en tanto que se obtiene una red con interfaces de bajo costo en las cuales el CPU realiza las mismas tareas que en las redes que utilizan interfaces altamente inteligentes. La clave de dicho rendimiento está en que se ha buscado utilizar componentes "inteligentes" existentes en el computador y que dispongan de mucho tiempo ocioso.

EL DISEÑO DE ANNETTE

En Annette se ha utilizado, como controlador de la red, componentes ya existentes en la mayoría de los computadores, como son el controlador de discos y el manejador de acceso directo a memoria (DMA). El uso de dichos componentes presenta las siguientes ventajas:

1) El procesador central del computador no se ve sobrecargado por el uso de la red.

2) El controlador de discos y el manejador de DMA son dos elementos capaces de realizar las funciones propias de un controlador de red, como son la serialización de los datos, la comprobación de errores, y la generación de

formatos compatibles para transmisión y recepción de mensajes.

3) Debido a que estos elementos se encuentran ociosos la mayor parte de su tiempo, pueden ser destinados a "escuchar" si llega algún mensaje.

4) Permite conectar gran variedad de equipos que utilizan controladores de discos compatibles.

5) La interfaz física agregada puede ser una tarjeta independiente que se conecta al computador como si fuera una unidad de disco flexible adicional y que por otra parte posee una conexión a la red.

6) Dicha interfaz sólo necesita realizar tareas sencillas, que se explicarán a continuación, por lo cual su costo es muy bajo.

Desde un punto de vista lógico, la interfaz a la red se encarga de que la dupla DMA - Controlador de Discos vea la red como si fuera una unidad de discos adicional (Fig. 1). Por dicho motivo, existirá una correspondencia directa entre las operaciones de escritura y lectura de discos (a bajo nivel), con la transmisión y recepción de paquetes en la red:

- La escritura de un sector con su formato apropiado en el disco virtual "red", corresponderá a una transmisión de un paquete.
- La lectura de un sector del disco virtual "red" corresponderá a la recepción de un paquete.
- Se asocia un número de sector en la escritura al disco virtual "red" con una dirección de destinatario.

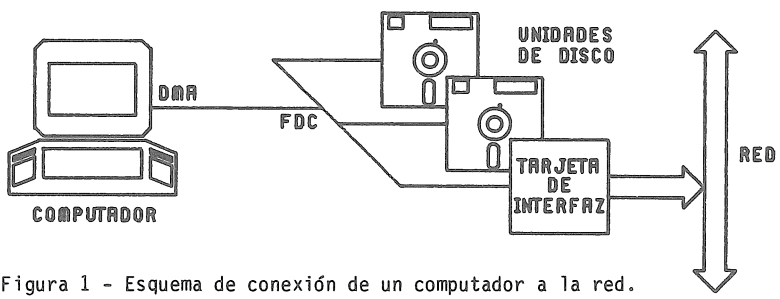


Figura 1 - Esquema de conexión de un computador a la red.

La comunicación entre máquinas se logra de la siguiente manera:

- Cada máquina posee una dirección única (asociada a un número de sector).
- Toda máquina se encuentra con su controlador de discos programado para leer el sector asociado a su propia dirección [11], y el manejador de DMA está pro-

gramado para llevar los datos leídos a posiciones de memoria específicas [12].

- La unidad de discos seleccionada es el disco virtual "red" por lo que el controlador de discos estará "escuchando" la red hasta que le llegue un paquete dirigido a él.

- Quien desea transmitir cambia la programación del controlador de discos y del manejador de DMA para que escriban en disco. (Para enviar un paquete se escribe el sector asociado a la dirección del destinatario, usando el modo "write track" del controlador de discos [11]).

- El destinatario recibe el paquete, ya que éste posee el formato del sector asociado a su dirección que el controlador de discos está esperando. El controlador de discos receptor realiza la comprobación de errores y pasa los datos recibidos al manejador de DMA, quien a su vez los transfiere a memoria. Al finalizar, el controlador de discos da una interrupción al CPU.

El método de acceso utilizado es de tipo CSMA/CD (Carrier Sense Multiple Access with Collision Detection), el cual es utilizado exitosamente por la red Ethernet [1]. En breve, es un método de acceso distribuido a un canal único compartido. Quien desea transmitir, si ve el canal libre puede hacerlo. En caso de que más de un computador intente transmitir ocurre una "colisión". Ésta es detectada por la interfaz, que constantemente compara lo que envía al canal con lo que recibe de éste. De ocurrir una colisión, cada máquina involucrada se retira por un tiempo aleatorio antes de intentar una retransmisión.

La topología de la red es tipo bus: el canal de comunicación pasa por todas las máquinas, y éstas se conectan a través de un corto cable de interfaz.

La topología y el control de acceso utilizado hacen que la red sea fácilmente reconfigurable, ya que el protocolo no depende del número de máquinas, y una máquina puede incorporarse o retirarse físicamente de la red simplemente conectándose o desconectándose del canal común.

La forma de propagación escogida por el medio de transmisión es de tipo "wired-or": el canal de transmisión se encuentra a un voltaje estable, y quien quiere transmitir lo lleva a 0 Volts. Esto se logra utilizando un medio de transmisión acoplado [8,9], con una fuente de voltaje continuo acoplada en un extremo, y una impedancia de terminación en el otro (Fig. 2). Cada computador, en su interfaz a la red, posee un comparador de alta impedancia de entrada para escuchar, y un transistor en estado de corte para transmitir (Fig. 3). Estos dos componentes se conectan a través de un cable corto al canal de comunicación, y como su impedancia es de varios órdenes de magnitud superior a la del medio,

puede considerarse que las interfaces al escuchar prácticamente no presentan desacople al medio de transmisión. La transmisión se realiza saturando el transistor de salida, el cual lleva a 0 Volts el voltaje del canal. El pulso se propaga en los dos sentidos hasta llegar a los dos extremos del canal donde es absorbido por las impedancias de acoplamiento. Se han hecho pruebas con par de cables telefónico y con coaxial standard de televisión (75 ohms), en ambos casos utilizando medio acoplado. La escogencia entre estos medios dependerá de la distancia máxima deseada de transmisión.

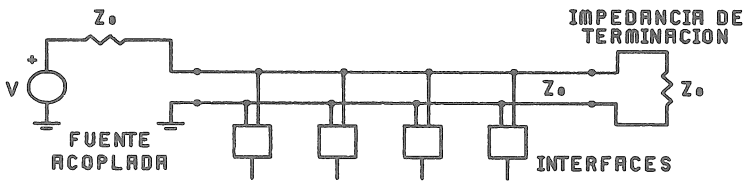


Figura 2 - Esquema eléctrico de la red.

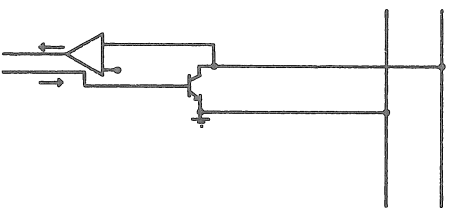


Figura 3 - Esquema de conexión eléctrica de una interfaz a la red.

La forma de codificación de los bits transmitidos es la misma utilizada por el controlador de discos, a nivel lógico [11]. La señal generada por el controlador de discos es llevada a niveles de voltaje apropiados (0 lógico es el voltaje estable del canal, y 1 lógico es 0 Volts o el voltaje de saturación).

La velocidad de transmisión es entonces la velocidad de escritura en discos. Las pruebas realizadas se han hecho con controladores de densidad sencilla, obteniéndose 250 K bits por segundo. De acuerdo al uso real de redes en funcionamiento [7] esta velocidad parece ser aceptable para gran variedad de aplicaciones.

La interfaz física es de bajo costo porque debe realizar tareas simples:

- Convertir niveles de voltaje

- Detectar colisiones (dicha detección en forma simplificada es una comparación entre lo que se escribe al canal y lo que se escucha).
- Simular el funcionamiento de una unidad de discos para el computador, por lo cual debe generar en forma apropiada los pulsos de "index" que corresponden al paso del agujero del disco flexible.

El formato de paquete corresponde con el formato utilizado por el controlador de discos [11], y en la figura 4 puede verse la correspondencia entre el significado que posee cada campo en la escritura de discos y en la transmisión de datos.

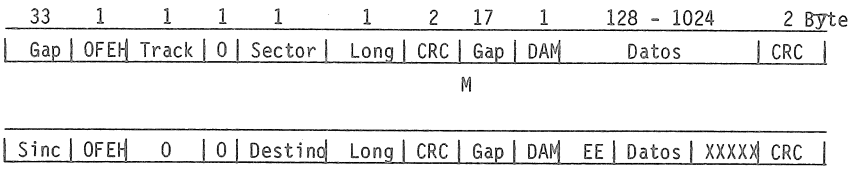


Figura 4 - a) Formato de escritura en discos. b) Formato de un paquete.

El identificador de sector en la escritura de discos es usado como identificador de destino y de longitud del paquete (comienzo del paquete en la figura 4 hasta el punto M). Este identificador es interpretado por el controlador de discos receptor, quien realiza las comprobaciones de error (corrobora el CRC) y pasa al computador el campo de "Datos" solamente.

Los primeros bytes del campo de datos (EE) se utilizan para:
1) Indicar el origen del paquete.
2) Indicar la longitud real de los datos (el campo de datos puede ser de varios tamaños posibles [11], pero puede estarse enviando menos datos que el tamaño completo).
3) Llevar a cabo el protocolo de enlace.

El protocolo de enlace se refiere al conjunto de normas a seguir, relativo a retransmisiones de paquetes, para proveer un canal de comunicación "libre de errores" [6]. El protocolo aquí utilizado es una variante del "Alternating Bit" [5], y cuenta con un reconocimiento ("ACK") inmediato de paquetes. Es importante tener un ACK en este tipo de red, ya que un paquete podría perderse si

el controlador de discos está ocupado realizando operaciones de lectura o escritura de discos.

En breve, el protocolo se desarrolla de la siguiente forma:

- Cada paquete lleva un bit de secuencia, el cual alterna cada vez que se transmite un nuevo paquete, pero se mantiene en caso de retransmisiones.
- Cuando una máquina recibe un paquete, envía inmediatamente un corto paquete (ACK) que avisa al origen la recepción correcta del mismo.
- Si el bit de secuencia del paquete recibido, corresponde con el esperado, el paquete se acepta como paquete nuevo.
- Si el bit de secuencia no corresponde, el paquete se bota por tratarse de un duplicado.
- Por su parte, el transmisor retransmite un paquete cuando no recibe el ack correspondiente.

El envío del ACK se hace en forma inmediata al recibir un paquete. Para evitar que el ACK pueda chocar con otro paquete, su envío toma lugar mientras todas las máquinas todavía están viendo el canal ocupado. El ACK inmediato permite que éste no deba llevar ninguna información. Su presencia indica bien recibido, mientras que su ausencia indica mal recibido o no recibido. Además, el ACK inmediato puede ser muy corto (un identificador de sector desde el punto de vista del controlador de discos), mientras que un paquete de ACK tendría que ser más largo y estaría sujeto a colisiones, lo que complicaría el protocolo.

Existe un paquete especial para inicializar el bit de secuencia en caso de fallas en la máquinas, o en caso de ser la primera vez que se envía un paquete a tal destino.

Por otra parte, cada paquete indica cuando es comienzo de mensaje y/o fin de mensaje, para que el receptor pueda reensamblar los mensajes, y sea capaz de reconocer un mensaje inconcluso.

ESTADO ACTUAL DEL PROYECTO

Se han diseñado y puesto a prueba las interfaces físicas en forma experimental lográndose comunicar máquinas con el método propuesto. Para las pruebas se ha usado el controlador de discos FDC-1771 [11], el cual utiliza formato compatible con IBM 3740.

Las rutinas que cumplen el protocolo de enlace descrito están a nivel de desarrollo e implementación bajo el sistema operativo CP/M, y se busca que no molesten los programas de aplicación de los usuarios. El diseño detallado del formato de los paquetes también se está finalizando y será publicado en [13].

GENERALIDAD DE ANNETTE

Annette es una arquitectura general de red. Las pruebas realizadas han sido hechas utilizando formato compatible con IBM, pero podría utilizarse cualquier otro. El único requerimiento es que todos los computadores tengan un controlador de disco compatible. Si un computador no lo posee, se le podría agregar una interfaz que tenga un controlador de disco. Se podría tener, además, pequeños microcomputadores dedicados al manejo de periféricos, que cuenten con un controlador de disco con el solo fin de conectarse a la red.

CONCLUSIONES

Se ha diseñado una red local de computadoras que permite conectar gran variedad de equipos que posean controladores de discos compatibles.

Las redes que existen en el mercado son una representación de una amplia gama de soluciones parciales a la relación de compromiso existente entre el deseo de que el CPU realice pocas tareas en el manejo de la red, y el deseo de que las interfaces sean económicas. No conocemos otra arquitectura que haya logrado ambas metas simultáneamente.

La red desarrollada presenta las siguientes ventajas:

1) Utiliza interfaces muy baratas.

2) No sobrecarga al CPU, debido a que casi todas las funciones las realizan componentes que disponen de mucho tiempo ocioso.

En desarrollos posteriores [13], se buscará evaluar la distancia máxima de operación de la red, así como el número de máquinas que ésta podría soportar. Actualmente, los cálculos están siendo realizados en base a una distancia de 500 metros por segmento de cable coaxial acoplado. El máximo número de computadores

que podrían conectarse a la red, podría estimarse de forma que ofrezca un servicio similar al que ofrece una red local comercial con alta carga. Consideramos que dicho número estará entre 25 y 50 máquinas.

Las posibles aplicaciones de la red dependerán de las evaluaciones antes mencionadas, pero en principio habrá diversas aplicaciones en las que podría ser utilizada, como por ejemplo oficinas que deseen utilizar un correo electrónico o compartir recursos costosos, como impresoras de calidad o unidades de almacenamiento masivo.

REFERENCIAS:

1) "ETHERNET: DISTRIBUTED PACKET SWITCHING FOR LOCAL COMPUTER NETWORKS", R. M. Metcalfe y D. R. Boggs, Comm. ACM, July 1976.

2) "THE ETHERNET, A LOCAL AREA NETWORK: DATA LINK LAYER AND PHYSICAL LAYER SPECIFICATIONS, VERSION 1.0", Digital Equipment Corporation, Intel, Xerox, Sept. 1980.

3) "EVOLUTION OF THE ETHERNET LOCAL COMPUTER NETWORK", J. Shoch, Y. Dalal, D. Redell, R. Crane, IEEE Computer, August 1982.

4) "UNTANGLING LOCAL AREA NETWORKS", R. Parker, S. Shapiro, Computer Design, March 1983.

5) "COMPUTER NETWORKS", A. Tanenbaum, Prentice Hall, 1981.

6) "OSI REFERENCE MODEL - THE ISO MODEL FOR OPEN SYSTEMS INTERCONNECTION", H. Zimmermann, IEEE Trans. Comm., April 1980.

7) "MEASURED PERFORMANCE OF AN ETHERNET LOCAL NETWORK", J. F. Schoch y J. A. Hupp, Comm. ACM, Dec 1980.

8) "FOUNDATIONS FOR MICROWAVE ENGINEERING", R. E. Collin, Ed. Mc Graw Hill, 1966.

9) "PROPAGATION ON A TRANSMISION LINE", Application Note 62, Hewlett Packard.

10) "RAMA", Tesis de Maestría en Ingeniería Electrónica de O. Moreán, Tutor: O. Chang, Universidad Simón Bolivar, Caracas, Venezuela.

11) "INS 1771-1 FLOPPY DISK FORMATTER/CONTROLLER", National Semiconductors, April 1977.

12) "8257 PROGRAMMABLE DMA CONTROLLER", Intel Corporation, 1978.

13) "RED LOCAL ECONOMICA DE ALTO RENDIMIENTO PARA MICROCOMPUTADORES", Tesis de Maestría en Ciencias de la Computación a ser presentada en el presente año, G. Broner, Tutor: P. O'Callaghan.

M.F. VIEIRA*, J.N. CHAME**, A.M. MOREIRA***, L.F.G. SOARES****

SUMÁRIO

Ligações multiponto em uma rede local em barra comum apresentam algumas características especiais. A atenuação introduzida pelo meio de transmissão e as reflexões geradas nas ligações a este são fatores que podem impor limites ao tamanho da rede, em termos de comprimento máximo e número máximo de estações, e à taxa máxima de transmissão. Este artigo faz uma breve análise dos vários tipos de transceptores para redes locais em barra comum (CSMA-CD e "token bus") para depois descrever o transceptor da rede local REDPUC. São apresentados então os acréscimos que este deve sofrer para sua utilização em uma rede com protocolo CSMA-CD. Ao final uma versão simplificada e de baixo custo de um transceptor é apresentada.

ABSTRACT

Multipoint connections on a global bus local area network show some uniques characteristics. The attenuation introduced by the means of transmission and the reflections generated by the connections are points that may restrict the maximum transmission rate and the size of the network, in respect to the maximum length and the maximum number of station. For the CSMA-CD and Token Passing global bus local networks, this paper gives a brief analysis of some types of transceivers. The REDPUC local network's transceiver is fully described. Also, the additional features necessary to use this transceiver with a CSMA-CD network are presented. Finally, a low cost and simplified version of the transceiver is derived.

*Engenheiro Eletricista(FEJ/UDESC,1980),mestrando em Sistemas de Comp.; redes de computadores,sistemas para transmissão de voz.

**Engenheira Eletricista(PUC/RJ,1984), mestranda em Sist.de Comp; redes de computadores, teleprocessamento.

***Engenharia Eletricista(PUC/RJ,1984) redes de Comput.,Teleproc.

****Doutor em Informática (PUC/RJ, 1983), Engenheiro Eletricista (PUC/RJ, 1976); redes de computadores, modelos para especificação e validação de protocolos, modelos para análise de desempenho, sistemas para transmissão de voz.

Pontifícia Universidade Católica do RJ, Deptº de Engenharia Elétrica Rua Marques de São Vicente 225, 22453 - RJ - Tel: 274-9922 - R/334.

1. INTRODUÇÃO

O desenvolvimento de redes locais veio a possibilitar a construção de máquinas de arquitetura distribuída, trazendo como consequência melhor desempenho do sistema, e a viabilizar o partilhamento de recursos(discos de grande capacidade, impressoras rápidas, banco de dados, etc).

Redes locais se constituem em um conjunto de estações interligadas por um sistema de comunicação. A forma com que as diversas estações são interconectadas é definida pelo arranjo topológico da rede. Entre as topologias mais usuais se encontram as em estrela, em anel e em barra comum |1|.

Nas redes com topologia em estrela ou anel as ligações são feitas ponto a ponto enquanto que nas redes em barra comum as ligações são multiponto. As ligações multiponto apresentam algumas características especiais,devido ao fato de que todas as estações (nós) são ligadas a um único meio de transmissão.

A atenuação introduzida pelo meio de transmissão e as reflexões geradas nas ligações a este são fatores que podem impor limites ao tamanho da rede, em termos de comprimento máximo e número máximo de estações.

Em redes de médio custo/médio desempenho e alto custo/alto desempenho o meio de transmissão tipicamente utilizado é o cabo coaxial. Ligações no cabo coaxial geram descontinuidade de impedância, causando reflexões, devido à impedância não infinita apresentada pelo conector (transceptor).Assim,o transceptor deve apresentar uma alta impedância (idealmente impedância infinita) para o cabo coaxial, de forma que a sua ligação a este altere o mínimo possível as características de transmissão. Além disso , a disposição das conexões deve ser controlada de forma que as reflexões não se somem em fase a um valor significativo.

Um outro problema, a inserção ou a retirada de uma estação da rede, pode implicar na necessidade de parada total do sistema, em função da forma,de acoplamento do transceptor com o cabo,utilizada. A escolha para a realização do acoplamento deve ser feita tendo-se em vista as aplicações e necessidades da rede, a simplicidade desejada e o custo.

Este artigo descreve as características de transceptores para redes locais em barra comum, que utilizam como meio de comunicação o cabo coaxial.
Na seção 2 são descritos os tipos de transceptores para redes locais em barra comum e as principais diferenças entre os transceptores utilizados em redes com protocolo CSMA/CD e passagem de permissão.

Na seção 3 é feita uma descrição em blocos de um transceptor para uma rede local em barra comum com protocolo de acesso por passagem de permissão.

Na seção 4 são apresentados os acréscimos necessários ao transceptor descrito na seção anterior a fim de adaptá-lo à uma rede em barra comum com protocolo de acesso CSMA/CD.

Na última seção são apresentadas as conclusões e é proposta uma versão simplificada e de baixo custo para um transceptor a ser utilizado em redes com protocolo de acesso por passagem de permissão.

2. TRANSCEPTORES PARA REDES LOCAIS EM BARRA COMUM

O transceptor é o responsável pela ligação das estações ao meio de comunicação, e tem como função básica transmitir e receber sinais através deste meio.

Os transceptores podem estar localizados junto as estações ou não, e ambas as configurações apresentam vantagens e desvantagens.

A localização do transceptor fora da estação traz como principal vantagem uma flexibilidade maior para o sistema. As estações podem se situar afastadas do meio de comunicação, estando ligadas ao transceptor por uma linha de comunicação (por exemplo um par trançado). Assim, dentro de limites estabelecidos pelas características do meio de transmissão entre o transceptor e a estação, (como atenuação, por exemplo) esta pode estar localizada em qualquer ponto conveniente, distante da barra comum.

É recomendado que neste tipo de configuração o sinal seja enviado diferencialmente entre o transceptor e a estação, de forma a diminuir o ruído de modo comum. É usual a utilização de par trançado para a ligação ponto a ponto entre as estações da rede e o transceptor, uma vez que este meio de transmissão possui boa imunidade a ruído, uma ligação simples, além de ser o meio de transmissão de menor custo por unidade de comprimento.

A localização do transceptor junto a estação traz algumas simplificações. Além da eliminação da necessidade de um meio de transmissão entre a estação e o transceptor, este pode ser implementado de forma mais simples (por exemplo, um possível circuito de detecção de colisão não necessita enviar um sinal diferencialmente para a estação para garantir imunidade a ruído, podendo portanto ser menos complexo). Por outro lado a estação tem que estar localizada o mais próximo possível do meio de transmissão, para evitar problemas de reflexão e atenuação do sinal. Assim a simplici

dade e o menor custo desta configuração implicam em menor flexibilidade para o sistema, em relação à configuração anterior.

Para estas duas configurações a conexão do transceptor ao meio(cabo coaxial) pode ser realizada de mais de uma maneira.

Uma forma de conexão com o cabo coaxial é a utilização de conectores tipo T. Este tipo de ligação apresenta a desvantagem da necessidade de interrupção do funcionamento da rede para cada inserção de uma estação. Em compensação os conectores em T são facilmente encontrados e seu custo é relativamente baixo.

Uma alternativa que permite a inserção ou a retirada de uma estação na rede sem a necessidade de interromper o funcionamento do sistema é a utilização de um conector por pressão ("pressure tap"). Com este conector o acoplamento com o cabo coaxial é feito por meio de um "probe". Este "probe" consiste de uma "agulha" metálica que é inserida no cabo coaxial fazendo contato somente com o seu condutor central. Para impedir que a malha externa entre em contato com a "agulha", esta é revestida com material isolante, com exceção de suas extremidades.

Este método de conexão apresenta ainda a vantagem de realizar o acoplamento com um mínimo de descontinuidade de impedância no cabo, o que se torna tão mais necessário quanto maior for a taxa de transmissão utilizada.

Além das funções (de isolamento, impedância infinita, retirada/inserção fácil, etc) já mencionadas, os transceptores possuem outras funções que vão depender do protocolo de acesso à barra da rede, como a função especial de detecção de colisão.

Em redes em barra comum com acesso por passagem de permissão, a probabilidade de ocorrerem colisões é muito pequena. Assim alguns protocolos podem optar por tratar colisões como erros a serem detectados por um campo do CRC (código cíclico de redundância), como é o caso do protocolo utilizado pela REDPUC [2]. Isto torna o transceptor mais simples,uma vez que ele não precisa realizar a detecção de colisão.

Já em outras redes em barra comum, como por exemplo as que utilizam protocolo de acesso CSMA/CD, existe a necessidade da detecção de colisão,e o transceptor vai ter também esta função. Isto vai implicar na utilização nestas redes de um transceptor mais complexo e consequentemente de maior custo.

Detalhes sobre esta função de detecção de colisão do transceptor são apresentados na seção 4, onde se descrevem os acréscimos necessários pa-

ra realizá-la.

3. TRASCEPTOR PARA UMA REDE LOCAL EM BARRA COMUM COM ACESSO POR PASSAGEM DE PERMISSÃO

O diagrama em grandes blocos do transceptor para REDPUC |4| é apresentado na Fig. 1. Os níveis dos sinais tanto no cabo coaxial quanto no cabo do transceptor seguem as especificações ETHERNET |3|.

O bloco de recepção tem como função básica captar o sinal no meio de transmissão e converte-lo para os níveis de sinal necessários à sua aplicação no par trançado.

O bloco de transmissão é responsável pela conversão do sinal que chega pelo par trançado para os níveis de corrente exigidos à sua aplicação no cabo coaxial.

A utilização de um conversor DC/DC com transformador é necessária devido a necessidade de isolamento DC entre o terra do transceptor e o terra das estações, para evitar que as mesmas fiquem com os terras interligados através da malha externa do cabo coaxial |5|

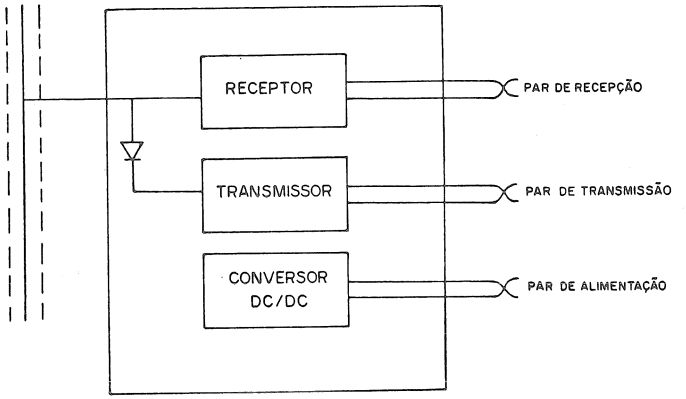


Fig. 1. Diagrama em Blocos do Transceptor REDPUC

Um par trançado é utilizado para a alimentaçao do transceptor, sendo esta enviada pela estação.

Na Fig. 2 é apresentado o diagrama em blocos do receptor. O estágio de entrada deve possuir alta impedância, de modo a garantir que o acréscimo de uma estação ao cabo coaxial não crie uma descontinuidade na sua impedância característica, evitando-se assim problemas de reflexão no cabo

coaxial.

O estágio conversor diferencial, configurado como "schmitt trigger", consiste de um receptor de linha ECL com saída diferencial e tem como função restaurar o sinal recebido do cabo e converte-lo para diferencial.

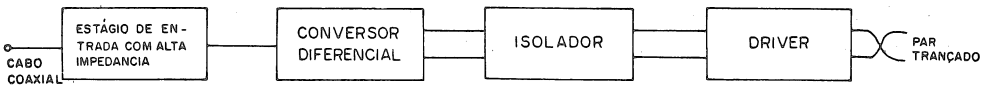


Fig. 2. Diagramas em Blocos do Receptor

Além do isolamento entre o terra do transceptor e o terra da estação também é necessário fazer um isolamento entre os sinais. As técnicas mais utilizadas são: isolamento a transformador e isolamento ótica. Transformadores de pulso possuem baixa impedância para sinais e alta impedância de modo comum para isolamento. Além disto, a sua implementação é relativamente barata. Isoladores óticos ainda não são muito utilizados devido ao custo e a pouca disponibilidade destes componentes para frequências de operação altas. O acoplamento capacitivo é uma técnica de isolamento mais barata que as duas anteriores, porém possui uma baixa rejeição de modo comum.

Após passar pelo bloco isolador o sinal vindo do cabo passa pelo driver, sendo então aplicado no par trançado nos níveis adequados. A implementação do driver pode ser feita utilizando-se um receptor de linha ECL. A utilização de lógica ECL deve-se à capacidade destes circuitos excitarem linhas de transmissão. O uso de linhas de transmissão permite manter a integridade do sinal sobre distâncias longas, possibilitando assim manter o transceptor distante da estação.

Na Fig. 3 é apresentado o diagrama em blocos do transmissor, o qual é composto por quatro blocos.

(1) O receptor de linha tem como função receber o sinal enviado pela estação através do par trançado e restaurá-lo, compensando assim a atenuação sofrida pelo sinal entre a estação e o transceptor. Um receptor de linha ECL com casamento de impedância na entrada pode ser utilizado para realizar esta função.

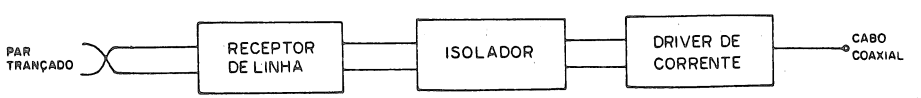


Fig. 3. Diagrama em Blocos do Transmissor

(2) Transformador de pulso cuja função é isolar os sinais entre o cabo coaxial e a estação, de forma idêntica ao do circuito receptor.

(3) O driver de corrente tem como função básica converter o sinal diferencial aplicado na sua entrada, para os níveis de corrente necessários a satisfazer as especificações de sinal no cabo coaxial. Com o objetivo de facilitar a explicação do funcionamento da implementação utilizada no transceptor da REDPUC, o mesmo será decomposto em quatro blocos: controle de chaveamento, conversor tensão-corrente, temporizador de transmissão e filtro de saída.

As fontes de corrente I_1 e I_2 são responsáveis pela geração do nível de corrente necessários no cabo coaxial. A posição das chaves CH1 e CH2 é comandada pelo controle de chaveamento, sendo dependente da polaridade do sinal aplicado na entrada do driver. Com as chaves nas posições indicadas na Fig. 4, a corrente no cabo coaxial é igual a I_2, fornecendo um nível de tensão de aproximadamente - 0,2V |3|. Com as chaves em posições invertidas, a corrente no cabo coaxial é igual a $I_1 + I_2$, fornecendo um nível de tensão no cabo de -1,8V aproximadamente.

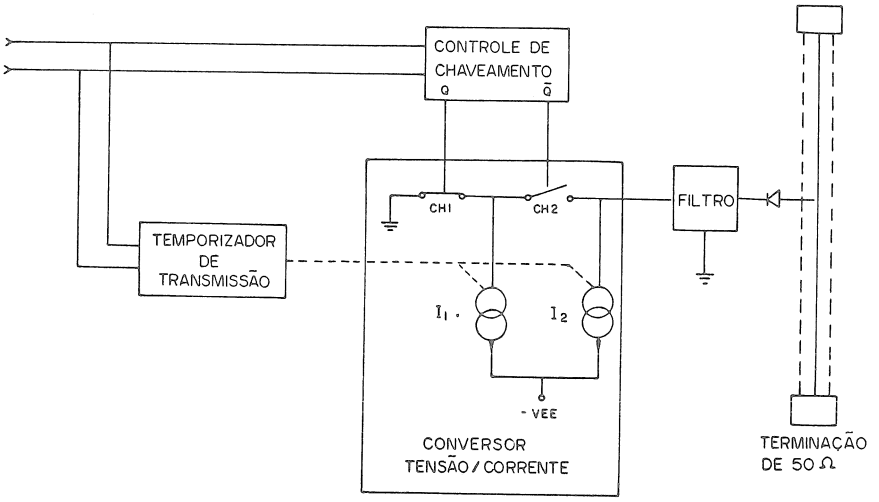


Fig. 4. Diagrama em Blocos do Driver de Corrente

O circuito de controle de chaveamento garante que a chave CH1 esteja fechada e CH2 aberta quando o transceptor é ligado, enquanto que o temporizador de transmissão mantém as fontes de corrente desligadas. Quando um sinal diferencial é aplicado na entrada deste estágio, o temporizador de transmissão é disparado ligando as fontes de corrente I_1 e I_2 durante um

tempo pré-estabelecido, enquanto que as chaves eletrônicas CH1 e CH2 ligam e desligam de acordo com a polaridade do sinal na entrada do estágio.

O temporizador tem como função permitir que a barra seja liberada após um determinado tempo, no caso de uma estação com defeito continuar transmitindo após ser esgotado o seu tempo de transmissão. Este tempo é igual ao tempo necessário para enviar um pacote acrescido de uma tolerância.

Quando uma transmissão é finalizada o controle de chaveamento garante que CH1 e CH2 retornem as condições iniciais; simultaneamente o temporizador de transmissão desliga as fontes de corrente I_1 e I_2.

(4) O filtro de saída tem como função reduzir as componentes de alta frequência adequando os tempos de subida e de descida (conforme a especificação ETHERNET de 25ns $\pm$ 5ns).

A conexão do transceptor ao cabo coaxial deve perturbar o mínimo possível as características da linha de transmissão. O diodo ligado entre o transceptor e o cabo coaxial tem como função reduzir a capacitância paralela de conexão. O transceptor deve ficar localizado o mais próximo possível do cabo coaxial. Conexões com comprimento maior que 3cm entre o cabo e a entrada do transceptor não são aceitáveis.

4. TRANSCEPTOR PARA REDES EM BARRA COMUM QUE UTILIZAM PROTOCOLO CSMA/CD.

Adicionando-se mais algumas funções ao transceptor REDPUC visto na seção anterior, o mesmo pode ser utilizado em redes onde é necessário detectar colisão. Na Fig. 5 é apresentado o diagrama em blocos desta implementação.

Este transceptor realiza funções idênticas as daquele apresentado anteriormente. Para realizar as funções adicionais necessárias foram acrescentados os circuitos detector de colisão e temporizador de colisão, este último encontrando-se junto com o bloco de transmissão. Ao ser detectada colisão pelos circuitos responsáveis por esta tarefa, a estação é avisada através de uma sinalização enviada pelo transceptor através do par de colisão

O temporizador de colisão garante que,quando ocorrer colisão de sinais no cabo coaxial, o transmissor seja desconectado do cabo somente após todas as outras estações terem detectado a mesma.

A detecção de colisão é feita por um circuito que verifica se o nível DC do sinal recebido é correspondente a níveis de transmissão sem colisão;em caso contrário, envia uma onda quadrada para a estação, avisando sua ocorrência. O bloco detector de colisão é composto de:

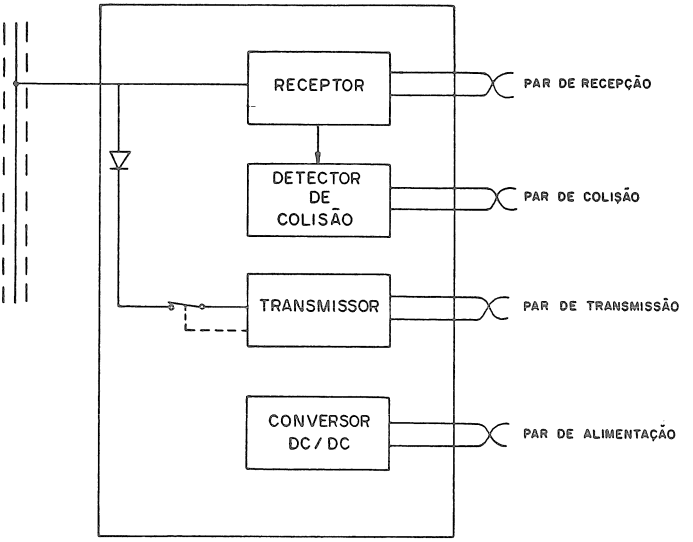


Fig. 5. Trasceptor para Redes com Protocolo CSMA/CD

(1) Um bloco comparador que detecta a presença de colisão no cabo coaxial.

(2) Um bloco gerador de onda quadrada com saída diferencial que oscila em 10MHz (por exemplo) enquanto persistir a colisão.

(3) Um transformador de pulso para isolamento dos sinais entre o transceptor e a estação.

(4) Um driver ECL que manda o sinal para a estação através do par trançado.

O temporizador de colisão tem como função desconectar o transmissor do cabo coaxial no caso de ocorrer colisão, após decorrido um determinado tempo. Este tempo deve ser suficiente de modo a garantir que mesmo a estação mais distante consiga detectar que ocorreu colisão. O circuito temporizador compara o nível dc no cabo coaxial com um valor de referência. No caso de ausência de colisão o nível dc no cabo, menor que o valor de referência, é constante devido as características da codificação manchester. Quando ocorre colisão o nível dc no cabo é alterado ultrapassando o valor de referência, condição esta que faz com que o transmissor seja desligado do cabo.

5. CONCLUSÕES

Uma versão simplificada de um transcepetor para fins experimentais ou nos

quais as estações estejam de preferência em um mesmo ambiente é apresenda na Fig. 6.

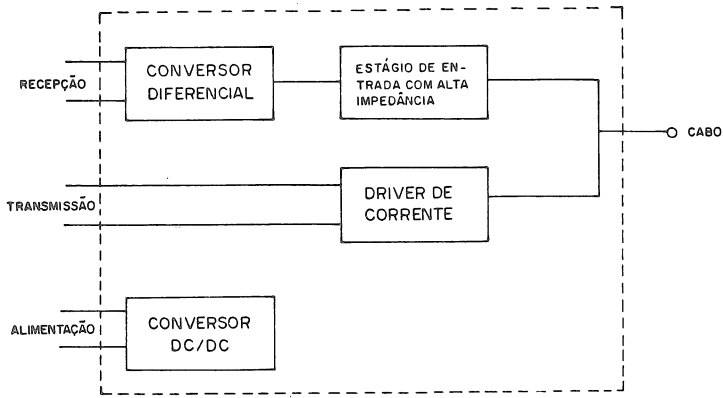


Fig. 6. Versão Simplificada para o Transformador

Os custos de conexão do transceptor com a estação podem ser eliminados colocando-se o transceptor dentro da própria estação. Com isto torna-se desnecessário o uso de conectores tanto do lado da estação quanto do lado do transceptor, assim como os pares trançados de recepção, transmissão e alimentação.

Os dois blocos do receptor, o estágio de entrada com alta impedância e o conversor diferencial possuem funções idênticas às apresentadas nas seções anteriores.Para a utilização deste transceptor é assumido que:

(1) A detecção de colisão é feita pela estação, como por exemplo erro de de CRC,ou ainda,a estação comparar os dados no cabo coaxial com aqueles que está transmitindo. A discordância nesta comparação indicaria colisão.

(2) A temporização de transmissão é feita somente na estação simplificando consideravelmente os circuitos do driver de corrente.

O circuito de temporização de colisão pode ou não ser acrescentado ao transceptor dependendo da implementação da estação utilizada assim como do protocolo de acesso ao meio de comunicação.

O conversor DC/DC é ainda necessário para manter o isolamento dc entre os terras das estações e a malha externa do cabo coaxial. Em aplicações onde as estações estão próximas e com um mesmo potencial de terra, pode-

se para fins experimentais, sob controle de laboratório, ligar o transceptor sem o conversor DC/DC.

O transceptor descrito no ítem 3 encontra-se em funcionamento nos laboratórios da PUC/RJ onde vêm sendo submetidos à testes e medidas. A referência (6) descreve minuciosamente o projeto deste transceptor e alguns testes já realizados.

5. BIBLIOGRAFIA

(1) Soares, Luiz Fernando Gomes, Projeto e Desenvolvimento de Protocolos para Redes Locais de Computadores. PUC/RJ Tese de Doutoramento - Departamento de Informática-Dez/83

(2) Soares, Luiz Fernando Gomes; Ierusalimsky, Roberto; Corrêa, Carlos Henrique C. e Barreto, Marcelo. Interface de Barra Microprogramada para Rede Local REDPUC. Anais do XVII Cong.Nac.Inf. - Nov/84

(3) Intel, Digital and Xerox The Ethernet. A Local Area Network Data Link Layer and Physical Layer Specifications, September 30, 1980

(4) Moreira, Anamaria Martins e Chame, Jacqueline Nóbrega, Transceptor para uma Rede Local do Tipo Difusão com Protocolo por Passagem de Permissão PUC/RJ - Trabalho Final de Curso - Departamento de Engenharia Elétrica - Dez/84

(5) Shoch, J.F., Dalal, Y.K e Redell, D.O. Evolution of Ethernet Local Computer Network, Computer, August, 1982

(6) Vieira, Marcus Fabio, Chame, Jacqueline Nóbrega. Um Transceptor para a Rede Local - REDPUC, Nota Técnica Departamento de Engenharia Elétrica PUC/RJ,abril/85

P.R. FOINA (*)

SUMÁRIO

Nesse trabalho apresentamos uma proposta de arquitetura não convencional baseada nas idéias de Dennis (DENNIS). A arquitetura apresentada é uma arquitetura controlada pelo fluxo de dados ("data-flow architecture") onde o único tipo de dados existentes é a cadeia de caracteres.

ABSTRACT

This paper presents a non-conventional architecture based on Dennis ideia (DENNIS). This architecture is a data-flow one where the character string is the only data structure.

(*) Bacharel em Física (UFSCar 76); Mestre em Ciência da Computação (ICMSC-USP 80); Doutor em Informática (PUC-RJ 85); Inteligência Artificial, Sistemas Especialistas, Processamento Gráfico, Computação de quinta geração; UFSCar-DCEs , caixa postal 676, CEP 13560 São Carlos SP, fone (0162) 71-8111 ramal 143.

1. Introdução

O desenvolvimento da informática sempre leva a uma constante busca de máquinas mais rápidas e, eventualemente, mais adequadas a determinadas aplicações. Estudos levados a cabo em meados dos anos 70, mostraram que as arquiteturas baseadas na proposta original de von Neumann (vonNEUMANN) estavam chegando próximos aos seus limites práticos. Essas análises, juntamente com a cada vez maior dificuldade em melhorar o desempenho das tecnologias de semicondutores, fizeram aparecer propostas de arquiteturas alternativas (BACKUS), (DENNIS), (TRELEAVE).

A proposta a ser aprsentada aquí corresponde a uma das arquiteturas em estudo pelo grupo de arquiteturas controladas pelos dados do Departamento de Computação e Estatística da Universidade Federal de são Carlos, com vistas a sua possível implementação na forma de protótipo.

2. Conceitos Básicos

Nas arquiteturas controladas pelos dados uma instruções é executada assim que os dados necessários à ela estejam disponíveis no computador, isto é, uma determinada instruções é executada tão logo os seus operandos estejam disponíveis, independentemente da sua posição no corpo do programa. Se, na nossa arquitetura, dispusermos de vários processadores então um grande número de instruções poderão ser executadas simultaneamente, desde que elas tenham todos os seus operandos disponíveis.

As arquiteturas controladas pelos dados são classificadas em duas categorias a) arquiteturas controladas pela disponibilidade de dados e b) arquiteturas controladas pela

demanda de dados.

As arquiteturas controladas pela disponibilidade de dados implementam, de forma completa, a filosofia de execução apresentada acima, ou seja, todas as instruções cujos dados estejam disponíveis são executadas assim que haja processadores livres.

As arquiteturas controlada pela demanda dos dados também implementam aquela filosofia de execução mas com uma pequena alteração, apenas são executadas as instruções realmente necessárias para a obtenção dos resultados do programa.

3. Características da Arquitetura Proposta

O desenvolvimento da nossa arquitetura balizou-se em função de certas premissas de projeto que discutiremos nessa seção.

A primeira premissa é a de que desenvolveremos uma arquitetura controlada pela disponibilidade de dados mas que minimizasse a execução de trechos desnecessários dos programas.

A segunda premissa foi adotar, como modelo básico de arquitetura controlada pelos dados, a proposta desenvolvida na Universidade de Manchester por Gurd e equipe (GURDa), (GURDb), (CATTO).

A terceira premissa é quanto ao tipo básico de dados a ser manipulado pela arquitetura. Foi adotado a cadeia de caracteres como estrutura de dados básica dessa proposta. Notem que um número é na verdade uma cadeia de dígitos e portanto não precisaremos nos preocupar com questão de precisão, representação, erros, etc. Da mesma forma uma texto é uma

cadeia de caracteres e como tal deve ser tratado.

A última premissa de projeto é quanto ao fato que, dentro de um programa qualquer, encontraremos certos trechos que serão executados obrigatoriamente em sequência. A nossa arquitetura deverá dispor de mecanismo para detectar esse fato e dar ao mesmo um tratamemto adequado de forma a não carregar os recursos internos do computador.

4. Descrição da Arquitetura

A nossa arquitetura é formada por tres grandes subsistemas a) Memória de Operandos (MO), b) Memória de Instruções (MI) e c) Sistema de Operadores (SO) interligados entre si por um sistema de comunicação em anel (AN). A figura 1 mostra essa arquitetura de maneira esquemática.

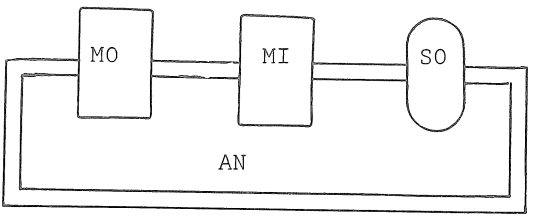


Figura 1. Esquema da Arquitetura Proposta

Um operando de uma operação binária, procura na MO o seu par. Caso ele não exista, esse operando é então armazenado (na própria MO) e fica esperando pelo par. Se o seu par já existir então ele é retirado da MO e ambos são montados juntos e

enviados para a MI.

Junto com os operandos está o indicador da instrução que os usará. Esse indicador não é o código da instrução mas apenas o número que essa instrução tem no grafo de fluxo correspondente ao programa em questão.

Na MI o indicador de instrução é substituido pelo código das instruções (pode haver mais do que uma) que irão efetivamente usar aqueles operandos. Além dos códigos de instruções, a MI anexa também os indicadores das instruções que usarão os resultados dessa operação. Ao chegar no Sistema de Operadores (SO), esse conjunto (operandos, código de instruções e indicador de destino) procura um operador que esteja disponível para a sua execução. O resultado dessa execução é um novo operando que será enviado para a MO afim de iniciar um novo ciclo.

5. Memória de Operandos

A memória de operandos armazena todos os operandos que são produzidos, ou introduzidos, durante a execução de uma programa. A MO é formada por um conjunto de Sistemas de Armazenamento de Listas (SAL) conforme mostra a figura 2. Em cada SAL temos uma estrutura de memória adequada ao armazenamento de listas encadeadas. Os operandos são formados por dois campos distintos a) indicador de instrução e b) informação. O campo indicador de instrução informa sobre qual a instrução que usará aquele operando. Essa indicação é feita através do número associado a cada instrução no grafo de fluxo

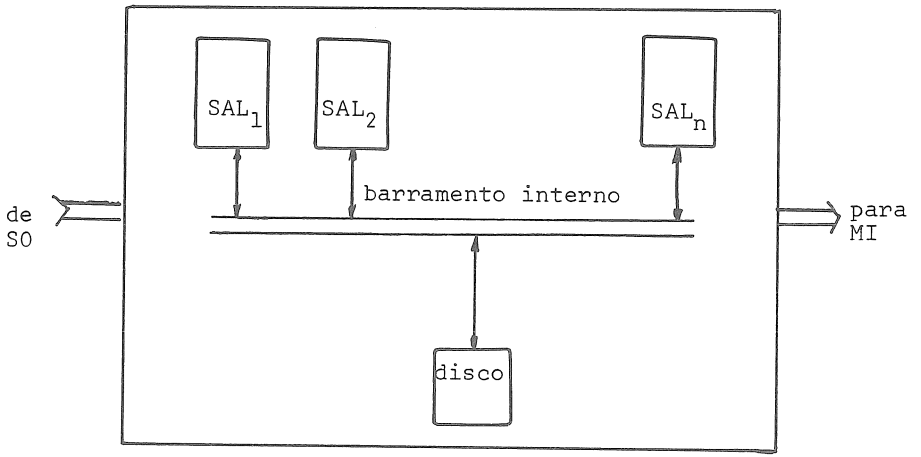


Figura 2. Sistema de Armazenamento de Listas

O compo de informação carrega o dado propriamente dito. Se o operando se destinar a uma operação aritmética teremos nesse campo uma lista de dígitos e, se o operando se destinar a uma operação sobre texto teremos então, uma cadeia de caracteres.

O indicador de instrução será usado para a alocação do operando dentro de um dos SAL existentes. Para cada SAL os indicadores de instrução formarão um diretório, acessado por rotinas de hash, que apontarão para o início do campo de informação correspondente.

O campo de informação ficará armazenado no SAL na forma de uma lista encadeada. Os ponteios dessa lista são gerados e gerenciados pelo próprio SAL sem que isso afete a informação

armazenada. A figura 3 mostra como é organizado um SAL.

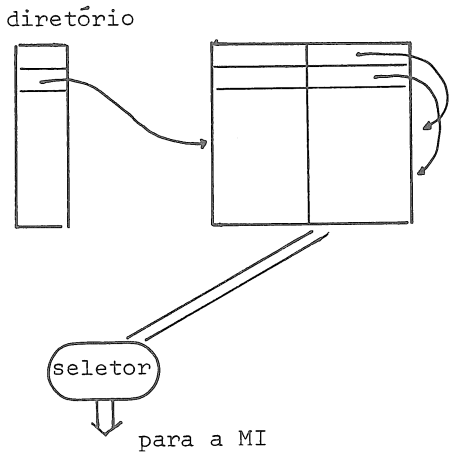


Figura 3. Estrutura da SAL

Em cada SAL existe uma unidade de seleção a qual se destina a extrair, de uma cadeia que esteja armazenada, uma sub-cadeia que será usada pela instrução que estara sendo executada. O recurso de equipar a SAL com uma unidade de extração de sub-estruturas torna-se necessário para evitar o trânsito de grandes estruturas de dados pela arquitetura.

6. Memória de Instrução

Similarmente à MO, a memória de instruções é formada por um conjunto de subsistemas de armazenamento denominados UAIs (Unidades de Armazenamento de Instruções). Cada UAI tem um diretório e uma memória estruturada em lista de forma similar à da SAL. A seleção da UAI correspondente e a busca da posição

onde está a instrução dentro da UAI é baseada no número da instrução que o operando carrega consigo.

O número da instrução corresponde a sua posição dentro do grafo de fluxo do programa. Esse identificador é substituido, na MI, pelo código da instrução correspondente. Um identificador pode corresponder a mais do que uma instrução e portanto pode-se executar um trem de instruções (que serão executadas necessariamente em sequência) de uma só vez minimizando a carga no anel. A figura 4 mostra a estrutura interna da MI e das UAIs.

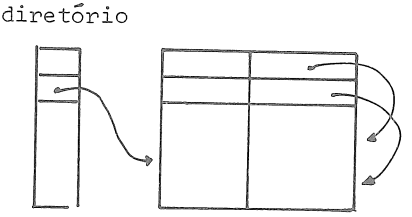


Figura 4. Estrutura das UAIs

7. Sistema de Operadores

O sistema de operadores é o orgão responsável pela efetiva execução das instruções. O sistema de operadores é formado por várias unidades de processamento cada qual capaz de executar qualquer uma das instruções do computador.

As unidades de processamento, denominadas de processadores (OP), estão interligadas entre si através de um barramento em anel. O anel tem duas portas de comunicação com o restante da arquitetura através das quais as operações são introduzidas e os resultados retirados do anel. A figura 5 mostra a estrutura desse anel.

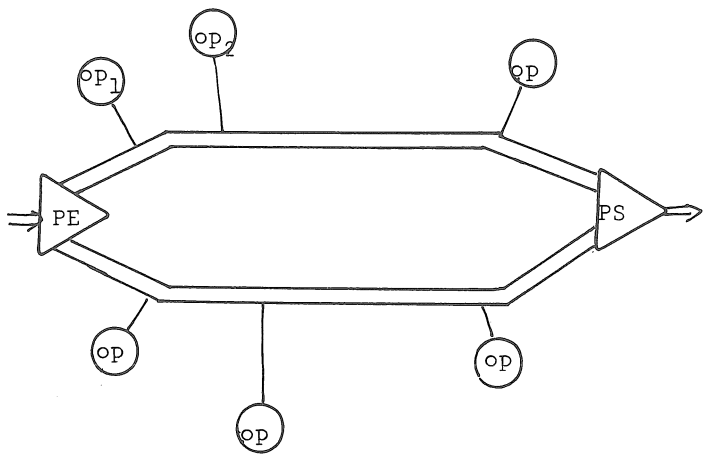


Figura 5. Estrutura do Sistema de Operadores (SO)

A rotina de operação do anel baseia-se na locação de todo o anel para permitir o envio de uma operação para um operador. Quando uma operação (instrução, operadores e destino) chega ao sistema de operadores ela é armazenada na porta de entrada (PE) do anel a espera da liberação de um operador. Quando um operador e o anel estiverem livres ele, o operador, faz

circular uma ficha com seu endereço. A porta de entrada do anel, ao detectar uma ficha de disponibilidade, retira essa ficha e começa a enviar, para o operador, a operação que tem armazenada.

A ligação entre a PE e o operador só é desfeita após a completa transmissão da operação. Enquanto uma ligação está feita nenhuma outra poderá ser conectada.

Ao acabar de executar uma operação, o operador espera a passagem de uma ficha indicativa de disponibilidade da porta de saida (PS) e do anel. Quando essa ficha é encontrada então o operador bloqueia o anel e faz a transmissão, para a PS, do resultado da operação. Novamente aquí, essa conecção só é desfeita após o término da transmissão.

Uma operação pode ter seu resultado enviado para várias operações destino, e nesse caso, o resultado é duplicado pelo próprio operador e enviado, um de cada vez, para cada uma das operações destinos. Para não bloquear demasiadamente o anel, nos casos de duplicação de resultados, após o envio de um resultado a conecção entre o operador e a PS é desfeita para permitir que outro operador assuma o controle do anel.

Exitem dois operadores especiais dedicados à aplicação específicas, eles são os operadores de entrada e de saída de informação. É através desses operadores que os dados de entrada são introduzidos no fluxo de execução de cada programa assim como são retirados os resultados finais. Os operadores de entrada e saída estão ligados aos dispositivos padrões de entrada e saída de dados no meio externo. A exitência de dois processadores distintos, um para entrada e outro para saída, deve-se à exigências de respostas rápidas feitas pelas aplicações em controle de processos e aquisição de dados. Com dois processadores independentes fica possível a entrada e

saída de informação simultaneamente.

8. Formato dos Dados

A figura 6 mostra a estrutura dos dados que transitam dentro da arquiteturas. O campo de operando é formado por uma cadeia de caracteres (de 8 bits). O campo de código de operação é também uma cadeia de caracteres onde cada caracter corresponde a uma instrução. Notem que a arquitetura disporá de apenas 256 instruções diferentes. O indicador de instrução é formado por 3 caracteres os quais poderão fazer referência a programas com até 32M instruções (nós do grafo de fluxo). Com a possibilidade de se agregar várias instruções em um único nó podemos ter na verdade aquele número aumentado.

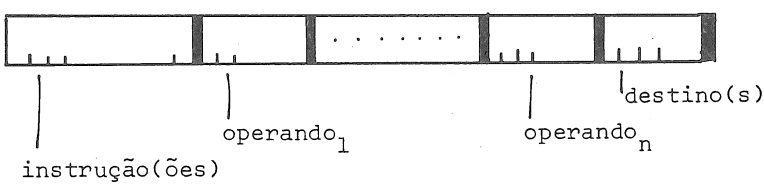


Figura 6. Formato dos Dados e Instruções

Cada operando carrega consigo, além dos valores propriamente ditos, um campo de descritores onde são indicados o nível de interação e a profundidade de chamada de procedimento. Outras informações contextuais podem ser anexadas

a esse campo de descritor. Certos subcampos são dedicados a aplicações específicas e são tratados diretamente pelos operadores (como é o caso dos indicadores de profundidade de chamada e de nível de interação). Os demais subcampos (tipicamente definidos pelos usuário) são igorados pelos operadores e copiados na resposta.

REFERÊNCIAS BIBLIOGRÁFICAS

CATTO, A.J., "Nondeterministic Programming in a Dataflow Environment", PhD thesis, University os Manchester, 1981.

BACKUS, J., "Can Programming Be Liberated From the von Newmam Style? Functional Style and this Algebra of Programs", CACM, vol. 21, n. 8, agosto de 1978.

DENNIS, J.B. et alli, "Data Flow Schema", in Lecture Notes in Computer Sciences, Springer Verlag, 1974.

FOINA, P.R., "LINDA - Linguagem para Descrição de Algoritmos Paralelos", artigo técnico interno do epartamento de Computação e Estatística da UFSCar, 1985.

_____ "Arquitetura e um Processador Funcional Usando uma Rede de Microprocessadores Dedicados", anais do VIII SEMISH, Florianópolis, julho de 1981. (b)

GURD, J.R., MATSON, I., GLAUENT, J.R.W., "A Data Driven System for a High Speed Parallel Computing", Computer Design, vol. 19, n. 6 e 7, junho/julho de 1980.

GURD, J. & WATSON, I., "A Multilayered Data Flow Computer Architecture", artigo técnico do Departamento de Computação da Universidade de Manchester, março de 1980.

TRELEAVEN, P.C., "Data-Driven and Demand-Driven Computer Architecture", ACM Computing Surveys, vol. 14, n 1, março 1982.

von NEUMANN, J. et alli, "Preliminary Discussion of the Logical Design of an Eletronic Computing Instrument", in TAUB, A.H., "Colleted Works of John von Newmam", vol. 5, pp. 34-39, MacMillan, Conpany, NY, 1968

ARQUITECTURA DE BUS DE MEMORIA CONTROLADA POR DEPENDENCIAS DE TAREAS

M. Nussbaum *

A. Cipriano **

RESUMEN

Se presenta un modelo MIMD basado en la idea de arquitecturas controladas por dependencias de datos. El modelo consta de una red que no requiere conmutación si no $O(n^2)$ memorias, donde n es el número de subsistemas que forman el sistema. Se caracteriza por permitir una ganancia de velocidad de procesamiento lineal con el número de procesadores, lo que es posible debido a que el sistema es controlado por un sistema operativo distribuido.

ABSTRACT

An MIMD architecture based on a dependence driven modell is presented. It consists on a network with no switching requirements but $O(n^2)$ memories, where n is the number of subsistems that conform the system. Its main caracteristic is a linear speed up of processing with the number of processors, which is possible through a distributed operating system, that controls the configuration.

* Ingeniero Civil Electricista (PUC 1980); M.Sc. Information and Computer Science (GA TECH, 1984). Profesor Escuela de Ingeniería, Departamento Ciencia de la Computación (143), P. Universidad Católica de Chile, Casilla 6177, Santiago, Chile.

** Ingeniero Civil Electricista (UCH 1973); Magister en Ingeniería Eléctrica (UCH 1974), Doktor Ingenieur (TU München, 1981); Profesor Escuela de Ingeniería, Departamento de Ingeniería Eléctrica (101), P. Universidad Católica de Chile, Casilla 6177, Santiago, Chile.

1 INTRODUCCION

La disponibilidad de microprocesadores de muy bajo costo y de muy alta capacidad de procesamiento hace posible hoy dia concebir supercomputadores basados en un gran número de estos dispositivos. Tales configuraciones, que conforman un sistema de multiprocesamiento del tipo MIMD (multiple instruction multiple data), hacen uso del paralelismo inherente a un programa, a objeto de conseguir que éste sea ejecutado en el menor tiempo posible.

Existen diferentes alternativas para implementar una máquina MIMD [1]. En este trabajo se presenta una arquitectura controlada por dependencias [2],[3],[4],[5] que incorpora nuevos elementos que le permiten un comportamiento lineal en su relación ganancia de velocidad versus número de procesadores.

En la sección 2 del trabajo se describe como presentar un programa como un grafo de tareas y la manera de procesarlo en forma paralela. En la sección 3 se describe el modelo de arquitectura propuesto y se finaliza con algunas conclusiones en la sección 4.

2 PROCESAMIENTO PARALELO DE UN PROGRAMA ESCRITO EN LENGUAJE DE ALTO NIVEL REPRESENTADO POR UN GRAFO DE DEPENDENCIAS

2.1 Descripción de un grafo

Un programa puede representarse como un grafo, cuyas unidades básicas se denominan nodos funcionales. Un nodo funcional es un segmento de programa que conforma una unidad que es difícil de paralelizar, como son relaciones recurrentes, condiciones indentadas, etc.

Para simplificar, supongamos que un nodo funcional es una subrutina de FORTRAN o un procedimiento o función de PASCAL. Un nodo tendrá así como entrada toda los datos que requiere para ser ejecutado, y su salida será el conjunto de datos que genere.

La figura 1 muestra un conjunto de procedimientos escritos en PASCAL, cuya representación como nodos funcionales se ilustra en la figura 2. El grafo de tareas de la figura 3, especifica el conjunto de todas las diferentes posibilidades de flujo entre los nodos funcionales del ejemplo de la figura 1.

Se observa que el flujo de ejecución puede ser alterado por toma de decisiones, como por ejemplo "repeat until" del nodo 0, el cual es una decisión que depende del booleano D. Cuando el resultado del loop es verdadero para D, la ejecución lleva a salir del loop; en caso contrario la ejecución permanece en el loop.

Se destaca también que el resultado de la ejecución de un nodo produce nuevos datos que pueden liberar más de un nodo, cuya ejecución puede así realizarse en paralelo. Por ejemplo, en el loop del nodo 0 es posible ejecutar dos ramas simultáneamente, las líneas (1) y (2) forman una de las ramas, mientras (3) y (4) forman la segunda. Además, la lectura de la variable F y la ejecución del nodo 2 con parámetro F pueden realizarse en paralelo con la ejecución del loop, dado que no existe interacción entre las variables.

2.2 Procesamiento paralelo

Un sistema de multiprocesamiento ideal es aquel en que la ganancia de velocidad crece linealmente con el número de procesadores conectados al sistema. Esto sólo es posible si la distribución de tareas ("schedulling") no se transforma en un cuello de botella al crecer el número de procesadores del sistema. Para esto es conveniente que el "scheduling" sea distribuido. Por otra parte, la ganancia de velocidad también se degrada por el costo computacional que corresponde a la comunicación requerida para el intercambio de datos.

En algunos casos es muy simple determinar la ganancia de velocidad que se obtiene en un sistema de multiproceso con "scheduling" distribuido, lo que permite comparar dos modelos de arquitectura. Supóngase que en un sistema de multiprocesamiento cada procesador ejecuta "scheduling", comunicación entre procesadores y ejecución de código, y que en un segundo sistema de multiprocesamiento existen dos procesadores independientes; uno para "scheduling" y comunicación y otro para ejecución de código. A fin de realizar la comparación entre ambas arquitecturas, empleando el programa ilustrado en figura 1, es necesario definir los siguientes términos:

Ts : Tiempo de ejecución serial total en un sistema de un sólo procesador.
Tpo : Tiempo de ejecución paralelo total, considerando que cada procesador ejecuta código, comunicación y "scheduling".
Tp : Tiempo de ejecución paralelo total, con procesadores independientes para ejecución de código y, "scheduling" y comunicación.
To : Tiempo de "scheduling" y comunicación para una tarea.
T1,T2 : Tiempo requerido para ejecutar los nodos 1 y 2 del ejemplo, respectivamente.
Tr : Tiempo necesario para leer datos de entrada.
n : Número de iteraciones en el loop de la figura.
S0 : Ganancia de velocidad que se obtiene con cada procesador realizando ejecución de código y, "scheduling" y comunicación.
Sp : Ganancia de velocidad que se obtiene con procesadores independientes para ejecución de código y, "scheduling" y comunicación.

Así se obtiene para el ejemplo anterior:

$$Ts = Tr + T1 + T2 + n*(Tr + 2*T1 + 2*T2)$$
$$Tpo = n*(3*To + Tr + T1 + T2)$$
$$Tp = n*(Tr + T1 + T2)$$

De aquí pueden determinarse las ganancias Spo y Sp:

$$Spo = Ts/Tpo = (Tr + T1 + T2 + n*(Tr + 2*T1 + 2*T2))/n*(3*To+Tr+T1+T2)$$

y

$$Sp = Ts/Tp = (Tr + T1 + T2 + n(Tr + 2* T1 + 2*T2))/(n*(Tr+T1+T2))$$

Puede observarse que para n grande que

Spo tiende a $(Tr + 2*T1 + 2*T2)/(3To + Tr + T1 + T2)$

y

Sp tiede a $(Tr + 2*T1 + 2*T2)/(Tr + T1 + T2)$

Finalmente, podríamos suponer para efectos de comparación que todos los tiempos T son iguales, obteniendose así :

Spo = 5/6 < 1
Sp = 5/3 > 1

Del análisis anterior se puede observar que al tener procesadores que ejecutan las tareas de ejecución de código, "scheduling" y comunicación puede obtenerse un comportamiento inferior con el sistema de multiproceso que con un sólo procesador serial, debido al "overhead" introducido. Por otro lado, al independizar la ejecución de código por un lado y "scheduling" y comunicación por otro, se observa que se aprovechan las características paralelas inherentes al algoritmo, obteniéndose en el ejemplo un mejoramiento de un 67%.

3 DESCRIPCIÓN DE LA ARQUITECTURA Y SU OPERACIÓN

3.1 Comunicación entre procesadores

Para solucionar el problema de canalizar la información entre procesadores se propone una red que ocupa $O(N^2)$ memorias, sin requerimientos de conmutación, donde N es el número de subsistemas, con cada subsistema constituido por dos procesadores, cuyas funciones se describen posteriormnete; dado que el costo de la memoria es bajo y en el futuro tenderá a bajar más aún, sistemas con Gigabytes de memoria comienzan a ser una realidad y los requerimientos anteriores no son excesivos.

A cada subsistema se asocian N buses de datos. De ellos, uno es bus de salida y los otros N-1 son buses de entrada correspondientes a los buses de salida de los N-1 subsistemas restantes. Además, cada subsistema tiene una memoria conectada a cada bus de entrada, la que cumple el papel de buffer. De esta manera cada subsistema tiene acceso directo a toda la información del sistema, eliminando los conflictos potenciales que pueden ocurrir al accesar los datos que requiere. Para obtener tal condición, cada subsistema envía a través de su propio bus las variables que ha procesado a todos los otros subsistemas, sin interrumpir la actividad de ninguno de ellos. Dado que todos los subsistemas tienen una unidad de memoria conectada a los N-1 buses de entrada (memoria de transferencia), los subsistemas obtienen la información que requieren directamente desde ésta sin alterar la ejecución normal de los otros subsistemas y sin necesidad de conmutación. Así se eliminan los conflictos de comunicación entre subsistemas, pues sólo un subsistema ocupa cada bus. Se observa que se ha obtenido un modelo de arquitectura en el cual la distribución de información se realiza mediante el proceso de direccionamiento, y no por una compleja red de conmutación. Esto es posible predefiniendo areas de memoria para las N memorias de transferencia. La única restricción es que los subsistemas requieren de un gran espacio de direccionamiento, lo que es en todo caso alcanzable con la tecnología actual de microprocesadores.

En el capítulo anterior se dedujo que el tiempo necesario para realizar el "scheduling" puede originar que un sistema de multiprocesamiento sea menos efectivo que un monoprocesador. En cambio, un modelo de arquitectura en que procesadores independientes realizan el trabajo del sistema operativo ("scheduling" y comunicación entre procesadores) y el trabajo de ejecución de código, permite obtener siempre tiempos de ejecución menores o iguales que con un sistema monoprocesador, pues los procesadores de código se encuentran activos el 100% del tiempo.

Para obtener la condición anterior, cada subsistema requiere de una memoria local accesable por los dos procesadores: el procesador de "scheduling" y comunicación (procesador S), y el procesador de código (procesador P). Se necesita además dividir la memoria local en dos bancos de memoria independientes; uno sobre el cual trabaje el procesador S, y otro sobre el cual trabaje el procesador P. Una vez que P finaliza su labor de procesar un cierto nodo, lo comunica a S, cambia de banco de memoria y comienza a ejecutar la tarea que S le había preparado con anterioridad. S vuelve a realizar su labor (a describirse mas abajo), pero empleando ahora el otro banco de memoria.

3.2 Distribución de tareas

El problema básico del "scheduling" o distribución de tareas, consiste esencialmente en la determinación de los nodos que están disponibles para ser ejcutados y la asignación de éstos a los procesadores que se encuentren sin trabajo. Esta labor es posible de realizar en forma distribuida, según el algoritmo cíclico que se describe a continuación.

Para poder realizar la asignación de tareas, se enumeran los procesadores y a los nodos disponibles se les asigna un criterio de prioridad, que puede basarse en el tiempo de espera, el número de nodos que libera una vez ejecutado o bién en otros criterios. La determinación de cuál de los nodos es conveniente de ser ejecutado por un subsistema dado no es sin embargo trivial, por el hecho que simultáneamente puede haber un grán número de procesadores S, buscando que nodo procesar próximamente. Este punto es sin duda uno de los mas críticos originados por la determinación distribuida de las tareas que se asignan.

Ya que gran parte de esta labor distribuida se duplicaría, es ventajoso contemplar un subsistema dedicado exclusivamente a esta labor común. Se le denominará subsistema de coordinación C y consistirá en cuatro procesadores C0, C1, C2 y C3.

El subsistema de coordinación C inicia una fase de "scheduling" con la lectura, desde su memoria de transferencia, de que procesadores están libres. El subsistema ordena y envía de vuelta a los procesadores S, mandando la lista sobre el bus dedicado para este fin, ocupando para esto el procesador dedicado C1 (ver figura 5).

También es necesario determinar qué nodos están disponibles para ser procesados, tarea que lleva a cabo el procesador dedicado C2. En figura 4, se observa que un nodo que fue liberado por un nodo predecesor no necesariamente es un nodo disponible por el hecho que un nodo puede tener muchos predecesores. Empleando la información sobre que nodos ya fueron procesados, obtenida por el procesador C2 de las memorias de transferencia de los procesadores, y el grafo del programa que se encuentra residente en la memoria local, se determinan los nodos que estan disponibles para su procesamiento. Estos nodos son ingresados a una lista que es ordenada por C3. La obtención de la lista ordenada de nodos se realiza así a base de una configuración "pipeline".

El procesador dedicado C0, realiza faenas de sincronización, tanto del pipeline (procesadores C2 y C3), como de los procesadores C3 y C1, sincronizando así la transferencia de amabas listas ordenadas. C0 tiene también la labor de ir eliminando de la lista de nodos disponibles de C2, aquellos nodos que ya fueron asignados en el ciclo respectivo.

Cabe destacar que la frecuencia con que este proceso de asignación se puede repetir esta dado por el máximo de tiempo requerido por los procesadores C1, C2 y C3, con lo que se obtiene una arquitectura síncrona de período variable (figura 6). Este intervalo de

tiempo dependerá del número de nodos y el número de subsistemas. El Número de nodos es un parametro dependiente del programa en ejecución, mientras que el número de subsistemas dependerá de la configuración. El tiempo máximo debido al número de subsistemas, dependerá del tiempo necesario en ordenar la lista de subsistemas que se encuentran buscando un nodo. Este tiempo es O(nlogn). Dado que n < 10000 se tiene que O(nlogn) = O(c*N) y O(c*N) = O(n) con lo que es posible obtener una ganacia lineal en la velocidad de procesamiento con el número de procesadores.

Así la labor de los procesadores S, a partir del momento en que algún S es informado que el procesador P asociado ha finalizado el procesamiento de la labor previamente asignada, queda reducido a:

1. Enviar la solicitud de requerimiento de nodo para procesar e indicar que el nodo correspondiente ha sido procesado.
2. Enviar los resultados obtenidos por el procesador P asociado.
3. Cargar en su memoria local las listas ordenadas de nodos y procesadores enviadas por C1 y C3 respectivamente.
4. Asociar ambas listas y elegir el nodo que le corresponde.
5. Cargar el código y los datos del nodo a procesar por su procesador P asociado en su memoria local compartida.

Cabe destacar que en el esquema anterior no es posible que ocurran "deadlocks", dado que ningún procesador P puede estar esperando por los datos de un nodo aún no finalizado de ejecutar, puesto que nodos que están esperando por datos no son considerados disponibles.

4 CONCLUSIONES

Se ha propuesto un modelo de arquitectura que aprovecha el paralelismo inherente a un programa, maximizando la concurrencia presente en él. Dado que el "scheduling" se lleva a cabo en forma distribuida, la configuración tiene la característica de poder crecer tanto como sea necesario. Se obtiene así un aumento lineal de la ganancia de velocidad respecto al número de procesadores presentes, restringido a la capacidad del algoritmo para ser procesado en forma paralela. Es importante destacar que este proceso distribuido no introduce "overhead", pues cada subsistema consta de un procesador dedicado a labores de sistema operativo y un segundo procesador para la ejecución de código.

Esta arquitectura fue concebida para alcanzar máxima velocidad, en tanto que la eficiencia no fue considerada del todo. Analizando este factor, se observa que el procesamiento de código representa a lo más el 50% de la capacidad total de procesamiento del sistema, dado que uno de los dos procesadores de cada subsistema se dedica exclusivamente al "scheduling" y comunicación. Sin embargo, considerando el estado actual de la tecnología, este punto no es de gran importancia, en la medida que lo fue en decadas pasadas cuando el sistema operavo tenía la mision de optimizar los costosos recursos del hardware.

Empleando procesadores convencionales de 16 bits (por ejemplo el Motorola 68000, el cual puede considerarse como una máquina de un MIPS), en una configuración de 100 subsistemas se podrían alcanzar velocidades de 100 MIPS y así capacidades de procesamiento similares a las de un supercomputador a un costo mucho menor. Además, hay que considerar que la velocidad indicada corresponde a procesamiento de código puro, puesto que el sistema operativo constituye un proceso concurrente y no implica en consecuencia "overhead".

Un inconveniente en la implementación de la arquitectura es la gran cantidad de buses requeridos. Cada procesador posee su propio bus, el que puede tener 40 o más líneas. Si se considera una configuración como la descrita en el párrafo anterior, se tendrían mas de 4000 líneas. Este punto puede solucionarse utilizando comunicación serial, con velocidades de transferencia de uno o dos órdenes de magnitud superior, lo que es alcanzable con la tecnología actual. Así se decrementa el número de líneas requeridas en la misma proporción.

La arquitectura propuesta constituye una arquitectura dinamicamente adaptable [5], [6]. Este tipo de arquitectura presenta las siguientes características:

1. Los recursos pueder reconfigurados a su mínima expresión.Esto significa que el sistema de mutiproceso con N procesadores, se puede reconfigurar por software para que hayan M (1 <= M <= N) grupos independientes trabajando. En este caso el número de programas concurrentes procesados simultáneamente es maximizado por el sistema, lo que permite la transformación de la máquina, de una máquina que maximiza "performance" a una gran máquina de "time-sharing".

2. Los recursos pueden ser conmutados entre diferentes tipos de procesadores (procesadores de I/O, máquinas pipeline, etc.), de tal manera que a cada procesador se asigne la tarea para la cual está mejor preparado. Así, en el ejemplo anterior se podría introducir un procesador de I/O que realice tareas de lectura.

REFERENCIAS

1. M.J. Flynn, "Some Computer Organizations and their Effectivness", IEEE Transactions on Computers, vol C-21, 1972, pp 948-960.

2. D.D. Gajski, D.J. Kuck and D.A. Padua, "Dependence Driven Computation", Proc. COMPCON, Spring 1981, pp 168-172.

3. R.G. Babb, "Parallel Processing with Large Grain Data Flow Techniques", Computer, Vol 17, Nº7, July 1984, pp 55-61.

4. J.L.A. Hughes, "Implementing Control-Flow Structures in Data Flow Programs", Proc. COMPCON, Spring 1982, pp 87-90.

5. S.R. Vegdahl, "A Survey of Prposed Architectures for the Execution of Functional Languages", IEEE Transactions on Computers, vol C-33, No.12, December 1984, pp 1050-1071.

6. C.R. Vick, S.P. Kartashev, and Steven I. Karatashev, "Adaptable Architectures for Supersystems", Computer, Vol 13 Nº11, November 1980, pp 17-35.

7. S.I.Kartashev and S.P. Karatashev, "Problems of Designing Supersystems with Dynamic Architectures", IEEE Transactions on Computers, Vol C-29, No. 12, December 1980, pp 1114-1132.

```
PROCEDURE Node_0 (var A: ARRAY [ 1..10] of integer;
                  var E:  boolean);
var B,C,F: Array [ 1..10] of integer;
      D: Boolean;
begin
   repeat read(A,B,C);
            Node_1(A,B,D);          (1)
            Node_2(A);              (2)
            Node_2(B);              (3)
            Node_1(B,B,E)           (4)
   until D;
   read(F);
   Node_2(F);
   Node_1(A,F,D);
end;

PROCEDURE Node_1(var A: ARRAY[ 1..10] OF integer;
                        B: ARRAY[ 1..10] OF integer;
                    var C: boolean);
var I: integer;
begin
    C:= FALSE;
    FOR I:= 2 TO 10
      do begin
            A[I]:= A[I-1] + B[I];
             IF A[I] < 100 then C:= true
         end;
end;

PROCEDURE Node_2(var A: Integer);
var I: integer;
begin
    FOR I:= 2 to 10
      DO A[I]:= (A[I] + A[I - 1]) / 2;
end;
```

FIGURA 2 Tres procedimientos escritos en Pascal, utilizados como ejemplos de nodos funcionales.

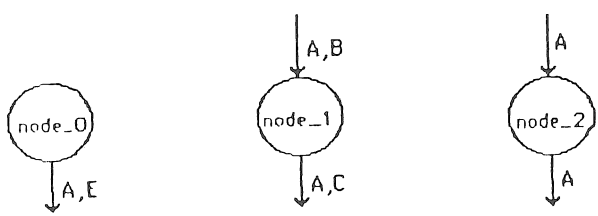


Figura 2 Nodos funcionales de los procedimientos de figura 1.

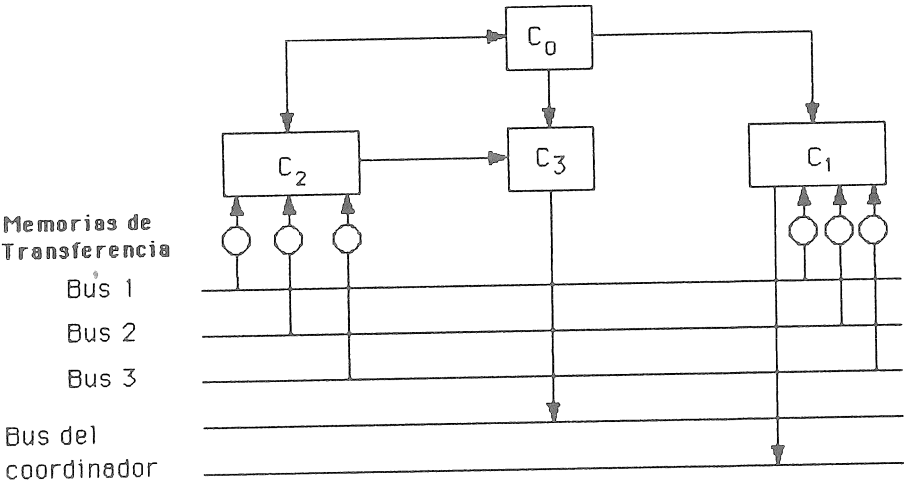


Fig. 5 Subsistema Coordinador.

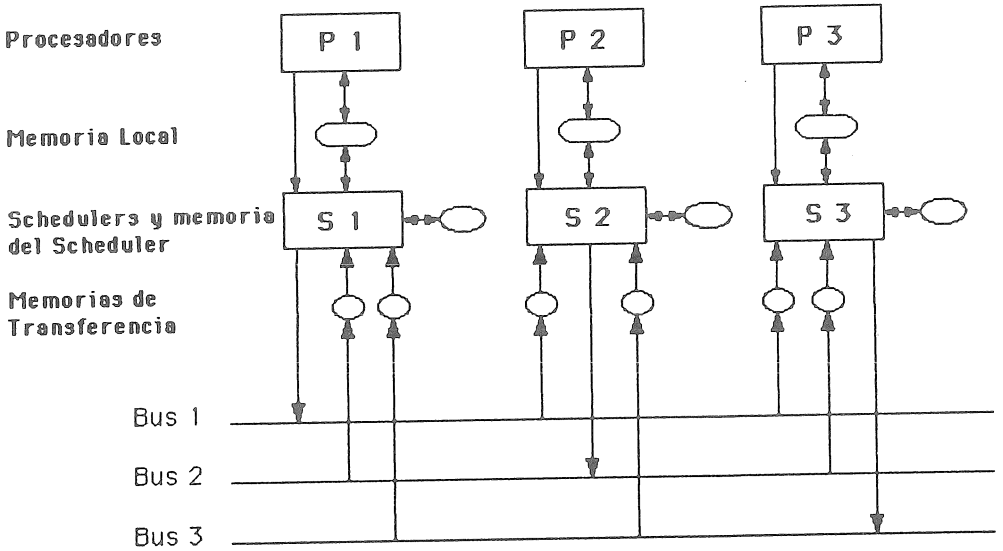


Fig. 6 Sistema de multiprocesamiento con tres subsistemas de dos procesadores cada uno.

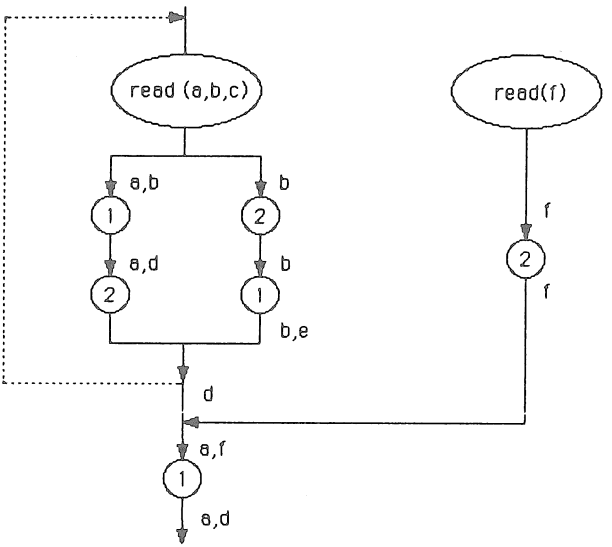


Fig. 3 Grafo de dependencias de tareas, para el ejemplo de la figura 1.

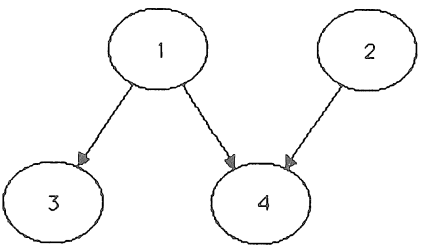


Fig. 4 Dependencia entre nodos.

A ARQUITETURA DOS SUPERCOMPUTADORES

C. L. AMORIM *

SUMÁRIO

A arquitetura dos processadores vetoriais pipelined (PVPs), popularmente referidos como supercomputadores, é colocada em perspectiva através de uma análise de sua estrutura e da interação de seus elementos básicos: 'pipelines' aritméticos, memória principal, memórias locais, 'pipelines' de instrução e instruções vetoriais. Os processadores vetoriais da Cray Research (CRAY1, CRAY X-MP), Control Data (CYBER 205) e Fujitsu (VP-200), atuais representantes do estado-da-arte são utilizados para consubstanciar a análise.

Os fatores que influenciam o desempenho desta classe de arquitetura assim como as principais linhas de pesquisa e desenvolvimento são apresentadas.

ABSTRACT

The architecture of pipelined vector processors(PVPs), usually referred to as supercomputers, is placed in perspective based on an analysis of the structure and interaction between its basic elements: arithmetic pipelines, main memory, local memories, instruction pipelines and vector instructions. The state-of-art vector processors from Cray Research (CRAY1, CRAY X-MP), Control Data(CYBER 205) and Fujitsu (VP-200) are used to illustrate and support the analysis.

The factors which influence the performance of such a class of processor architecture as well as the main research and development areas are introduced.

* Engenheiro Eletricista (PUC/RJ-1974), MSc (COPPE-1979) PhD (Imperial College-1984); Arquitetura de processadores pipelined e paralelos: modelagem, simulação e avaliação de desempenho; Professor Adjunto da UFRJ; COPPE/Sistemas, UFRJ, Caixa Postal 68511, CEP 21941, Rio de Janeiro, RJ.

1. INTRODUÇÃO

Os supercomputadores que podem ser definidos como os computadores de maior desempenho numa determinada época, tiveram historicamente marcos bem distintos. Durante os anos 60, a competição entre a IBM e a Control Data produziram os excepcionais processadores escalares IBM 360/195 (uma síntese dos IBM 360/91 e IBM 360/85) e CDC 7600 (evolução do inovador CDC 6600). No início dos anos 70, sem a participação da IBM, surgiram os processadores vetoriais pipelined da Texas Instruments (TI ASC) e Control Data CDC STAR 100. Essa primeira geração de processadores vetoriais, com poucas máquinas produzidas, serviu para identificar principalmente os problemas relacionados com o processamento vetorial. Com o lançamento do CRAY1 pela Cray Research em 1976, iniciando a segunda geração dos processadores vetoriais |1|, o mercado dos supercomputadores recebeu um novo impulso (em 1982 ± 70 CRAY1 haviam sido vendidos). A Control Data a partir da experiência do STAR 100 desenvolveu o CYBER 203 e em 1981 produziu o primeiro CYBER 205 para competir com o CRAY1. Em 1984, a Fujitsu com os processadores vetoriais VP-100 e VP-200 entrou fortemente no mercado |2|.

Um fato ainda mais importante foi a recente criação pelo governo americano de vários centros espalhados pelo país dispondo de supercomputadores |3| |4|. Tal medida resultou da constatação de que tanto a qualidade da pesquisa americana como a como a competitividade de sua industria estará comprometida se não se fizer frente as políticas nacionais da Alemanha, França, Inglaterra e Japão que vêm estimulando intensamente a exploração do potencial dos supercomputadores, colocando-os disponíveis aos pesquisadores de suas universidades e centro de pesquisas.

Acredita-se que assim como os microcomputadores vêm causando nos últimos anos uma revolução nas industrias, os supercomputadores no outro extremo do espectro, irão causar nos próximos anos uma maior revolução ainda. Segundo o prêmio Nobel de física Ken Wilson |3|, o uso de computadores na simulação de protótipos e otimização de novos produtos e processos é geralmente o método existente mais barato e mais rápido. Acontece que são poucos os cientistas e engenheiros provenientes das universidades que sabem algo sobre computação em larga escala |5| de modo a poderem utilizá-los eficazmente assim como contribuirem com novas idéias para o desenvolvimento dos supercomputadores.

Apesar disso as maiores industrias têm se beneficiado da computação em larga escala. Em particular, as industrias petrolífera (no tratamento de dados geológicos, simulação do fluxo de petróleo nas jazidas), aeronaval e automobilística (simulação do desempenho de protótipos), eletrônica (projetos de chips a três dimensões) e energética (simulação de reatores nucleares e rêdes de potência).

Espera-se que nos próximos 10 anos, na medida em que a utilização da integração em larga escala como vem ocorrendo com os supercomputadores em desenvolvimento pela Fujitsu, Hitachi e Control Data, computadores de altíssimo desempenho estarão disponíveis a preços 10 a 20 vezes menores aos atuais e comparáveis aos superminis disponíveis

hoje no mercado como o VAX da DEC.

No Brasil devido ao número relativamente pequeno de laboratórios de computação, os quais ainda sofrem de pequena capacidade computacional, poder-se-ia pensar numa política mais efetiva na qual o governo, as industrias interessadas (por exemplo as já citadas) juntamente com os laboratórios das universidades poderiam atuar na área de forma integrada.

Ressaltada a relevância do assunto passemos ao objetivo do trabalho. Tecnicamente falando os supercomputadores CRAY1 (CRAY X-MP, sua versão mais nova), CYBER 205, Fujitsu VP-200 pertencem a classe dos processadores vetoriais 'pipelined' (PVP). Neste artigo, a estrutura dessa classe de processadores será analisada. Para tanto iremos concentrar a atenção nas propriedades e na interação entre os componentes básicos da arquitetura, ou sejam, a) 'pipelines' aritméticos, b) memória principal, c) memória local, d) 'pipelines' de instrução e as instruções vetoriais.

A sequência de apresentação é a seguinte. Após observar que os PVPs exibem um objetivo de desempenho comum, ou seja, os processadores são organizados para processarem eficientemente instruções vetoriais através do princípio de 'pipelining' sem penalisar o desempenho das instruções escalares, nós examinaremos operações vetoriais 'pipelined'. Em seguida, verifica-se que esta condição determinará os limites inferiores de desempenho dos outros componentes da arquitetura e assim passaremos a analisá-los.

2. PIPELINES ARITMÉTICOS

A necessidade de instruções vetoriais surge ao se efetuar operações repetidamente numa sequência de pares de operandos, por exemplo, numa adição de dois vetores (B & C) como representado no seguinte 'loop' FORTRAN:

```
      DO 1 I=1,N
1     A(I)=B(I)+C(I)
```

Este 'loop' quando traduzido para linguagem de máquina corresponderá a uma sequência de instruções escalares que definirão a variável de incremento do 'loop', carregamento dos operandos nos registradores, adição de operandos, armazenamento do resultado e teste de fim de 'loop'. Tais regulares e relativamente simples instruções são bem adequadas para uma implementação rápida e eficiente em 'hardware'. Por outro lado, a frequente ocorrência desses cálculos particularmente em aplicações científicas onde a manipulação de matrizes numéricas é uma operação comum, justificaram o desenvolvimento de duas classes de processadores que representam duas estratégias para solucionar o problema. Uma é o processador vetorial paralelo tal como o Burroughs Scientific Processor [6], uma evolução do ILLIAC IV que emprega um número relativamente grande de elementos processadores (EP). A cada par de operadores é destinado um EP e a operação é executada simultaneamente nos EPs. A outra classe corresponde aos processadores vetoriais 'pipelined' que utiliza um número pequeno de 'pipelines' aritméticos para realizar os cálculos através do princípio de 'pipelining'.

'Pipelining' |7| |8| é definido genericamente como a divisão de uma função a ser realizada em sub-funções que podem ser executadas automaticamente por 'hardware' dedicado. Dessa forma sucessivas execuções da função podem ser conduzidas pelas sub-funções operando por superposição ('overlapped'). Um exemplo de aplicação desse princípio é 'pipeline' aritmetico (PA).

Um PA básico aceita 2 operandos a cada ciclo de relógio T e produz 1 resultado C ciclos depois. O valor C depende do número de estágios que a operação requer. O desempenho de uma PA é geralmente medido em termos de seu 'throughput', ou seja, o número de resultados escalares produzido por uma unidade de tempo. Em PVPs, uma unidade comum é MFLOPS (milhões de operações de ponto flutuante por segundo) que para o PA descrito acima é dado por 1/T e que independe do número de estágios C.

As configurações típicas dos PVPs empregam mais do que um PA para obter maior desempenho. O CRAY1 (CRAY-X-MP, processador dual) como visto na figura 1, possue 12 unidades funcionais ou PAs unifuncionais onde cada PA só pode executar um tipo de operação. Seu ciclo de relógio é de 12.5 ns,(9.5 ns), com operandos de 64-bits, 'throughput' de 80 (105) MFLOPS por PA, estes poderão ser encadeados nas instruções vetoriais e multiplicar o desempenho do processador.

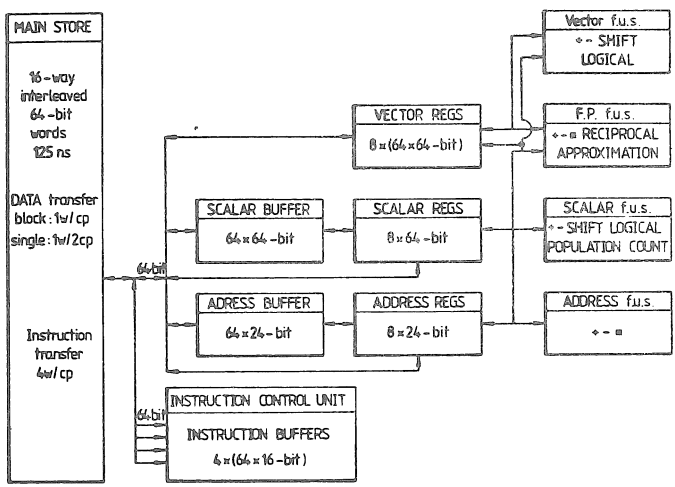


Figura 1: A arquitetura do CRAY1

No CYBER 205, representado na figura 2, os PAs (até 4) são multifuncionais estáticos ou seja, um PA só pode executar um tipo de operação de cada vez, mas pode ser configurado para realizar diferentes operações. O ciclo do relógio é de 20 ns, com operandos de 64-bits e 'throughput' de 50 MFLOPS por PA. No máximo duas instruções vetoriais podem ser encadeadas e assim dobrar o 'throughput' de um PA mas diferente de CRAY1 um dos

operandos dessas instruções tem de ser escalar.

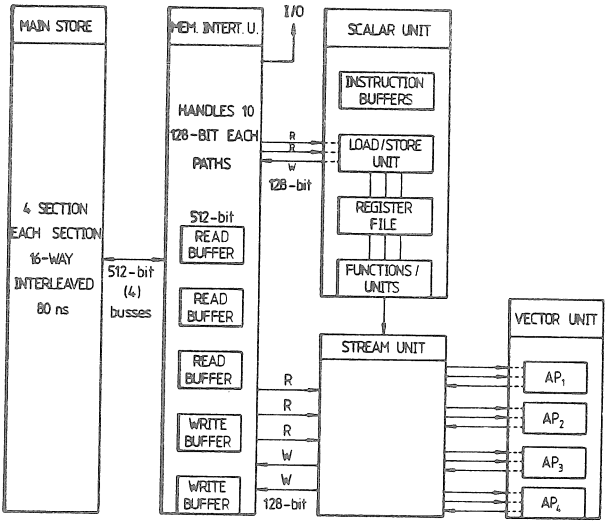


Figura 2: A arquitetura do CYBER 205

3. MEMÓRIA PRINCIPAL

Se o 'throughput' dos PAs forem considerados como o objetivo dos projetos dos processadores vetoriais, os requisitos da memória principal (MP) podem ser estimados. Para sustentar o fluxo máximo de processamento é necessário que a MP forneça 2 operandos e armazene 1 resultado a cada ciclo do relógio o que determina uma bandapassante da memória equivalente a três vezes (taxa de demanda) o 'throughput' do processador.

A memória principal do CRAY1 (CRAY X-MP) é composta de 'chips' de 1K bipolar, com 1 milhão de palavras, intercalada em 16 bancos com ciclo de 50 ns (38 ns) e bandapassante apenas de 80 MP/s (milhões de palavras de 64-bits por segundo). Este desbalanceamento entre a bandapassante de memória e o 'throughput' do PA foi corrigido no CRAY X-MP cuja bandapassante é de 420 MP/s com 4 barramentos independentes para aproveitar a intercalagem dos bancos e ainda suportar acessos de entrada e saida. Para aliviar o provável engarrafamento da memória principal, o CRAY1 dispõe de uma memória local formado por oito registradores vetoriais, cada um podendo armazenar até 64 elementos de 64-bits. Instruções vetoriais permitem transferir blocos de dados (vetores) entre as duas memórias.

A memória do CYBER 205 é composta de 'chips' de 4K bipolar, com ciclo de 80 ns e é dividida em 4 seções independentes, cada uma com 1M palavras de 64-bits. Cada seção é intercalada de 16 modos e acessada através de uma barra de 512-bits de largura. Cada seção é organizada em 16 pilhas de 8 bancos de memória, cada um com 16K palavras de 32-bits. A intercalagem é obtida pelo fornecimento por cada pilha de uma palavra de

32-bits em paralelo.

A interface de memória gerencia 4 barras, que servem, cada uma a uma única seção e organiza o despacho de pedidos de acesso vindo do resto do sistema. Estes pedidos são feitos através de 10 barras de dados, cada um com 128-bits de largura, dos quais 5 são dedicados ao processamento vetorial, 3 ao processamento escalar e 2 ao processamento de E/S. Os pedidos de acesso a memória são montados em 5 'buffers' de dados, cada um com 512-bits de largura dentro da interface. Cada seção fornece uma bandapassante de 400 MP/s e portanto pode satisfazer a taxa de demanda dos PAs.

É importante observar que a intercalagem da memória no CYBER 205 e no CRAY1 é implementada diferentemente. No primeiro, palavras de memória com endereços consecutivos poderão ser lidas num ciclo. No segundo, palavras com endereços regularmente espaçados (espaçamento de 0 a 15) poderão ser acessadas. Esta última forma de intercalagem é claramente mais flexível e oferece significativa vantagem para o processamento vetorial como será visto adiante.

4. MEMÓRIAS LOCAIS

Memórias locais (ML) permitem que se utilize eficientemente a bandapassante da memória principal e desempenham um papel importante nos PVPs como ilustrado pelos registradores vetoriais presentes no CRAY1. É interessante notar que o mecanismo de registradores vetoriais introduzido no CRAY1, foi ainda mais explorado no recente Fujitsu VP-200 [2] que dispõe de 8K palavras de 64-bits para configurar tais registradores, ou seja, variando o número de elementos, de 8 a 256 registradores vetoriais poderão ser definidos.

Outras formas especializadas de MLs, baseadas nas características de acesso a dados e instruções pelos programas, têm sido utilizadas. O processamento de instruções endereçadas sequencialmente promoveu o uso de 'buffers' de instruções (BI) que são capazes de guardar com antecedência varias instruções que eventualmente serão processadas, a menos que ocorra uma ramificação. Esta capacidade extra pode ser explorada eficientemente por 'pipelines' de instrução como veremos adiante. No CRAY1 existem 4 BIs, cada um dispõe de 64 palavras (uma palavra pode conter até 4 instruções). No CYBER 205, 64 palavras são utilizadas como BI (cada palavra pode conter até 2 instruções).

Uma propriedade dos acessos a operandos é que elas tendem a se agrupar num pequeno conjunto de segmentos de memória por um período de tempo e depois migrar para um outro conjunto e assim por diante. Esta propriedade tem sido explorada nos processadores escalares de alto desempenho através do uso de MLs denominadas cache de dados [9].

Nos PVPs, as instruções vetoriais com acesso independente a memória principal exarcebam o problema de coerência dos dados o que faz com que somente formas simples de cache de dados programáveis sejam utilizadas. Assim, no CRAY1 são utilizados 2 conjuntos de 64 elementos (registradores B e T) que servem de memória temporária por exemplo, para preservar os 16 registradores de trabalho (8-A e 8-S) entre chamadas de subrotinas de modo a evitar o acesso a memória principal. Na verdade, a impossibilidade de indexar

os registros (B e T) em tempo de execução (eles têm que ser explicitamente alocados durante a compilação) dificultam sua utilização. O CYBER 205 possue 272 registradores escalares de trabalho.

5. 'PIPELINES' DE INSTRUÇÃO

Para descrever e caracterizar o processamento de instruções nos PVPs é util considerar separadamente cada uma das fases do processamento, ou seja, despacho de instruções, processamento escalar e processamento vetorial.

Os 'pipelines' de instrução (PI) controlam o carregamento e o despacho das instruções. O tráfego de instruções entre os BIs e a memória principal é feita em blocos (técnica de 'lookahed' |10|) e utilizam eficientemente a bandapassante da memória: por exemplo no CRAY1 (4 barras de 64-bits, 320 MP/s) e o CYBER 205 (1 barra de 128-bits, 100 MP/s).

Instruções escalares e vetoriais são decodificadas e despachadas pela unidade de controle centralizada do PI. Assim que uma instrução é despachada, o 'pipeline' aritmético selecionado pelo PI assumirá o controle da execução. Isto permite que as instruções seguintes sejam processadas e entregues ao PA apropriado, aumentando o 'throughput' do processamento de instruções.

O processamento escalar tanto no CRAY1 como no CYBER 205 possue um conjunto dedicado de PAs. Isto permite que a cada ciclo uma instrução seja despachada e a execução de várias instruções superpostas. No entanto isto pode causar riscos ('hazards') entre as instruções na medida em que as operações compartilham recursos e necessitam de diferentes tempos de execução e podem não seguir a sequência predefinida e o resultado não ser o esperado. Para prevenir tal situação ocorra ambos PVPs empregam mecanismos de reserva sobre os recursos (exemplo: registradores, unidades funcionais) que estão sendo utilizados. Assim, por exemplo, instruções não serão despachadas de qualquer dos seus registros operandos tiver sido reservado.

O processamento vetorial no CYBER 205 é realizado nos múltiplos e independentes PAs de ponto flutuante, cada um deles contem 5 unidades funcionais que se comunicam através de uma via de dados comum. No CRAY1 existem 3 unidades funcionais vetoriais e o processamento vetorial ainda compartilha das 3 unidades funcionais de ponto flutuante utilizadas por operações escalares. A diferença significativa entre os dois PVPs é que enquanto no CYBER 205 os vetores operandos e resultado das operações vetoriais residem na memória, no CRAY1 eles se localizam nos registradores vetoriais. A consequência é que o tempo de inicialização ('start up') do processamento vetorial é significantemente maior no CYBER. Assim para pequenos vetores (menor do que 100 elementos) o CRAY1 apresenta melhor desempenho.

6. PROPRIEDADES DAS INSTRUÇÕES VETORIAIS

O repertório das instruções dos PVPs |11| (219 instruções no CYBER, e 120 no CRAY1 aproximadamente) compreendem instruções escalares e vetoriais. Os conjuntos de

instruções escalares são do tipo convencional de três endereços e não requerem uma melhor descrição. Ao contrário dessas, as instruções vetoriais apresentam propriedades únicas que valem a pena avaliar.

Instruções vetoriais têm o formato de três endereços que definem 2 vetores operandos e 1 vetor resultado. Informações vetoriais básicas consistem de: número de elementos, endereço inicial na memória e incremento para indicar os elementos seguintes. Nesse particular o CRAY1 é diferente do CYBER 205 e permite incrementos não unitários fazendo com que matrizes possam ser acessadas por linha, coluna, diagonais, etc... e utilizar em tais situações a bandapassante de memória eficazmente. Por exemplo:

(VL é um registrador de 6-bits que indica o comprimento dos vetores operandos no CRAY1)

Vi Ao,Ak transfere (VL) palavras para o registrador vetorial Vi a partir do endereço (Ao) incrementado por (Ak)

Em contraste com o CYBER que pode processar com uma única instrução vetorial vetores de até 64K elementos, o CRAY1 opera com vetores de no máximo 64 elementos. Para processar vetores maiores eles terão que ser divididos e operados em grupos de 64.

Além dos vetores densos (que exibem uma configuração de endereços regulares na memória) existe uma outra importante classe constituída pelos vetores esparsos nos quais um grande número de elementos são zeros. Para processar eficientemente esses vetores é necessário evitar o processamento de operações nos elementos nulos. A solução típica é associar um vetor de controle (formado de bits) a cada vetor que participará da operação. Um elemento do vetor é selecionado para processamento de acordo com o valor do bit correspondente do vetor de controle.

Operações envolvendo vetores de controle permitem vetores esparsos serem compactados, utilizados nesta forma em operações vetoriais e depois expandido para as formas originais. O CYBER 205 dispõe desta facilidade que também é empregada para superar o problema de endereçamento contíguo a memória pois qualquer configuração de 1's pode ser especificada. Porém a menos que a memória aceite tal 'string' de controle, palavras consecutivas serão acessadas de qualquer forma. No CRAY1 o registrador VM de 64-bits serve de forma mais rudimentar ao propósito do vetor de controle em algumas poucas operações de 'merge' dois exemplos são dados a seguir:

CRAY1: Vi Vj!Vk&VM (coloque em Vi o elemento de Vj ou Vk de acordo com o valor correspondente bit de VM.)

CYBER 205: MPYUV (A,X),(B,Y),C,Z (multiplique os vetores especificados por (A,X) e (B,Y) sob controle do vetor Z e coloque o resultado no vetor especificado por C e C+1)

A,B,C,X,Y,Z são registradores

(A,X): 48-bits do registrador A indica o endereço base do vetor

16-bits do registrador A indica o comprimento do vetor

16-bits do registrador X indica o deslocamento do endereço base

Existem ainda dois outros grupos de instruções vetoriais. O primeiro envolve a utilização de um vetor índice que indica quais os elementos de um vetor operando que irão participar da operação, como numa consulta indireta a uma tabela. Essas operações são referidas como REUNE ('gather') e DISPERSA ('scatter'). O outro grupo corresponde a operações aritméticas denominadas REDUÇÕES VETORIAIS que produzem um resultado escalar a partir de vetores operandos, por exemplo, o somatório dos elementos de um vetor e o produto interno de dois vetores. Somente o CYBER 205 dispõe desses grupos de instruções vetoriais.

7. CONSIDERAÇÕES SOBRE O DESEMPENHO DOS PVPs

Somente a partir do insucesso da primeira geração de PVPs diante dos processadores escalares de alto desempenho é que modelos analíticos foram utilizados para avaliar o impacto escalar nos PVPs |12| |13|. O fato é que os programas não podem ser completamente escritos utilizando instruções vetoriais e portanto o aumento do desempenho que pode ser obtido dependente não só da quantidade do processamento vetorial mais também da eficiência do processamento escalar.

Estudos envolvendo modelagem, simulação e avaliação de desempenho |14| |15| |16| |17| |18| e propostas de novas arquiteturas de PVPs |14| |19| têm surgido na literatura de modo crescente na medida em que o acesso ao PVPs pela comunidade científica tem sido ampliado.

Embora observado que a maioria das soluções dos problemas científicos e de engenharia envolve a manipulação de dados na forma matricial, outros fatores também influenciam a eficácia do processamento vetorial como um todo |20| |21|. Entre estes fatores destacam-se os problemas associados com o processo de conversão do 'software' desenvolvido para máquinas escalares para os processadores vetoriais. As soluções encontradas frequentemente se enquadram em duas categorias:

I - Desenvolvimento de extensões para a linguagem FORTRAN e de novas linguagens vetoriais de alto nível |22| |23|, que permitem ao usuário codificar diretamente sua aplicação sob a forma de sintaxe vetorial.

II - Desenvolvimento de compiladores vetoriais e técnicas de vetorização |24| que procuram recuperar a estrutura vetorial codificada na sintaxe escalar.

Outro fator importante é o esforço que se requer para reformular a física, a aproximação numérica e algorítmos para explorar a capacidade de paralelismo nos cálculos a serem executados no 'hardware' vetorial. Uma solução rotineira tem sido a de se desenvolver algorítmos vetoriais e paralelos |25| para serem utilizados na aplicação.

8. CONCLUSÕES

A presente análise dos supercomputadores permite se ter uma visão mais global e crítica dessa classe de arquitetura em contraste com as colocações mais específicas e descritivas encontradas na literatura.

Na opinião do autor, é imperativo o desenvolvimento mais intenso da computação científica em larga escala no Brasil, na medida em que as principais industrias nacionais atingem um estágio que necessita de tecnologias mais avançadas. Devido aos recursos econômicos exigidos é preciso uma política nacional que integre industrias, laboratórios de pesquisa e universidades para a utilização do potencial dos processadores de alto desempenho.

REFERÊNCIAS

|1| KOZDROWICKI, E. W., THEIS, D. J., "Second Generation of Vector Supercomputers",IEEE COMPUTER, nov 1980.

|2| MENDEZ, R., ORSZAG, S., "The Japanese Supercomputing Challenge", DATAMATION, vol 30 no 7, may 15 1984.

|3| DALLAIRE, G., "American Universities Need Greater Access to Supercomputers", CACM, vol 27, no 4, may 1984.

|4| "Access to Supercomputers: an NSF Perspective", entrevista com E. T. MAYES., CACM, vol 27, no 4, may 1984.

|5| RODRIGUE, G. et al., "Perspective on Large Scale Scientific Computations", COMPUTER, Tutorial Series, oct 1980.

|6| JENSEN, C. "Taking Another Approach to Supercomputers", DATAMATION, february, 1978.

|7| RAMAMOORTHY, C. V., LI, H. F., "Pipeline Architecture", ACM Computing Surveys, vol 9, no 1, march 1977.

|8| KOGGE, P. M., "The Architecture of Pipelined Computers", Mcgraw hill Book CO.,1981.

|9| SMITH, A. J., "Cache Memories", Computer Surveys, vol 14, no 3, sep 1982.

|10| KELLER, R. M., "Lookahead Processors", Computing Surveys, vol 7, no 4, dec 1975.

|11| HOCKNEY, R. W. & JESSOPE, C. R., "Parallel Computers", Adam Hilger ltd., Bristol 1981.

|12| RUDSINSKI, L., WORLTON, J., "The Impact of Scalar Performance on Vector and Parallel Processors", High Speed Computer and Algorithm Organization, Kuck, D. J. et al. (eds) Academic Press, NY, 1977.

|13| WIJK, H. C. VAN DER, "A Comparison of the CRAY1 and CDC 7600 Computer Systems", Mphil Thesis, Computing Dept., Imperial College, 1981.

|14| AMORIM, C. L., "Simulation Performance of a Class of Vector Processors", PhD Thesis, Computing Dept., Imperial College, 1984.

|15| LANG, D. E. et al., "A Modelling Approach and Design Tool For Pipelined Central Processors", IEEE Proc. Symp. Computer Architecture, 1979.

|16| PAUL, G., "Studies in Vector Architecture", Research Report, rc 9245, Computer Sciences Dept, IBM T. J. Watson RC, 1982.

|17| BASKETT, F. & KELLER, T. W., "An Evaluation of the CRAY1 Computer", High Speed Computer and Organization, Kuck et al. (eds.), Academic Press. 1977.

|18| SRINI, V. P., ASENJO, J. T., "Analysis of CRAYIs Architecture", the 10th. Annual Intern Symp. on Computer Architecture, SIGARCH, vol 11, no 3, jun 1983.

|19| SMITH, J. E., "Decoupled Access/Execute Computer Architecture", IEEE, 9th Annual Symp. on Computer Architecture, SIGARCH New, vol 10, no 3, apr 1982.

|20| WIRSCHING, J. E., KIRSHI, T., "Matching Machines and Problems", High Speed Computer and Algorithm Organization, Kuck, D.J. et al. (eds), Academic Press NY, 1977.

|21| CALAHAN, D. A., "Algorithmic and Architectural Issues Related to Vector Processors" AP 1027-776, Michigan Univ., Ann Arbor, System Eng. Lab., 1976.

|22| BASILI, V. R., KNIGHT, J. C., "A Language Design for Vector Machine",SIGPLAN notices, vol 10, no 3, March 1975.

|23| PERROT, R. H., "A Language for Array and Vector Processors", ACM Transaction on Programming Language and Systems, vol 1, no 2, Oct. 1979.

|24| AMORIM, C. L., "An Evaluation of Vectorization Techniques", MSc Thesis, Dept. of Computing, Imperial College, 1980.

|25| FONG, K. & JORDAN, T. L.,"Some Linear Algebraic Algorithms and their Performance on CRAY1", LA 6774,LASL,1977.

L. F. J. Maia - IFGW/UNICAMP
P. C. Bezerra - DEE-FEC/UNICAMP

SUMÁRIO

O trabalho propõe um método de reconhecimento de conceitos a partir de observações. Os conceitos são representados em um espaço conceitual, cujas coordenadas são parâmetros de livre definição pelo usuário. Cada conceito é caracterizado como uma região difusa no espaço conceitual; o conjunto de conceitos assim caracterizados se constitui em uma base de conhecimentos, e um processador de similaridade mede os graus de pertinência de cada observação aos conceitos presentes na base de conhecimentos.

O sistema forma a base de um projeto de pesquisa em desenvolvimento na Unicamp. Um protótipo foi implementado em um VAX-11/780, e uma versão operacional encontra-se em desenvolvimento para operar em micro-computador.

ABSTRACT

The paper proposes a method of concept recognition from observations. The concepts are represented in a conceptual space whose coordinates are parameters of free definition by the user. Each concept is characterized as a fuzzy region in the conceptual space; the set of concepts thus characterized forms a knowledge base, and a similarity processor measures the membership degrees of a given observation to the concepts in the knowledge base.

The system is a research project in development at Unicamp. A prototype was implemented in a VAX-11/780 and an operational version is being developped to operate in a micro-computer.

Universidade Estadual de Campinas
Cidade Universitária "Zeferino Vaz" - Barão Geraldo
13.100 - Campinas, SP

INTRODUÇÃO

Um processo de decisão consiste em definir ações que devem ser exercidas sobre um mundo físico, no sentido de modificá-lo de acordo com um dado objetivo.

O processo global de decisão pode ser decomposto em um processo de observação, através de sensores, que produz uma imagem mental do mundo real; um processo mental que produz, a partir da imagem mental, do objetivo e de uma base de conhecimentos armazenados em memória, uma prescrição de ação, e um processo de atuação que traduz a prescrição, através de atuadores, em uma ação sobre o mundo real.

O processo mental é, por sua complexidade, de difícil descrição formal e algorítmica, pois raramente é inteiramente consciente, e usa conceitos e processos vagos e incompletos. O mundo formal convencional, com variáveis precisas e algoritmos categóricos, não parece adequado para reproduzir os processos realizados no mundo mental.

Grande parte dos processos mentais podem ser modelados como operações de reconhecimento, onde uma observação é comparada com um conjunto de conceitos estruturados e armazenados em uma memória, sendo a observação reconhecida como representativa do conceito ou conceitos que lhe são mais similares.

O trabalho propõe um sistema especializável de reconhecimento de conceitos $S(X,Y,Z,O,C,R)$, constituido por um conjunto de objetos X, um espaço conceitual Y que tem por coordenadas os conceitos atômicos que podem ser definidos sobre os objetos, um conjunto de conceitos Z onde são classificados os objetos, uma operação de observação $O:X \rightarrow Y$ que associa cada objeto a um ponto no espaço Y, uma base de conhecimentos $C:Z \rightarrow Y$ que associa uma região difusa no espaço Y a cada conceito, e um processador de reconhecimento $R:X \twoheadrightarrow Z$ que determina um índice de similaridade do objeto aos conceitos definidos na base de conhecimentos.

CARACTERIZAÇÃO DE CONCEITOS

Um atributo é definido como uma função de variáveis ou aspectos mensuráveis ou avaliáveis do mundo real. Um atributo pode ser a altura de uma pessoa (medida em centímetros), o índice de esbeltez de um pilar (medido pelo quociente de sua altura pela seção transversal) ou a credibilidade de um político (avaliado por um grau de zero a dez).

Um conceito atômico j sobre um atributo i é uma função $y_j = F_j(x_i):R \rightarrow [0,1]$, que associa cada valor $x_i \in R$ do atributo i a um valor $y_j \in [0,1]$. $F_j(x_i)$ é a função de pertinência do atributo i ao conceito j, e y_j o índice de pertinência do atributo i, de valor x_i, ao conceito j.

O espaço conceitual Y é o espaço definido pelos conceitos atômicos y_j, onde serão caracterizados os conceitos complexos que completarão a base de conhecimentos.

Na implantação inicial do sistema foi colocada a restrição de permitir, na caracterização de conceitos atômicos, as funções:

$$C(x,a,b) = (2^{**}w) / (1+2^{**}w) \qquad \text{e}$$

$$P(x,a,b) = 1 / 2^{**}(w^{**}2) \qquad \text{, com } w = (x-a)/b$$

Como exemplo, sobre o atributo "altura medida em centimetros" podemos definir o conceito atômico "ALTO" com a função C (crescente) como <alto>=C(x,170,5), e o conceito atômico "MEDIANO" com a função P (pico) como <mediano> =P(x,170,10).

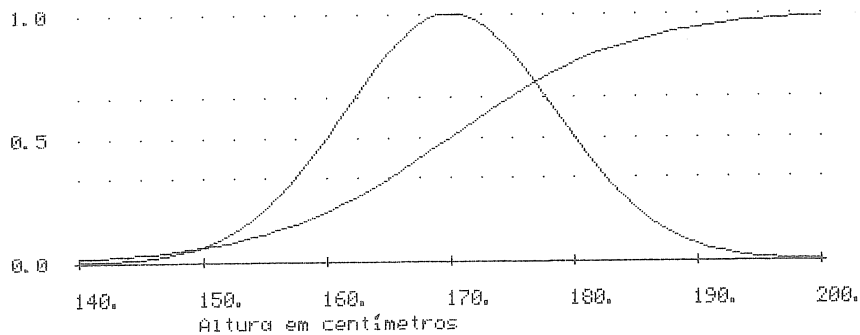


Fig. 1 - Conceitos "ALTO" e "MEDIANO"

Os parâmetros a e b definem, respectivamente, a posição e a largura da função, observando-se:

C (a-b) = 1/3	P (a-b) = 1/2
C (a) = 1/2	P (a) = 1
C (a+b) = 2/3	P (a+b) = 1/2

Definindo o operador <não> como:

$$\text{<não> } X = 1 - x$$

podemos definir as funções complementares das duas anteriores como:

$$D(x,a,b) = 1 - C(x,a,b) \qquad \text{e}$$

$$V(x,a,b) = 1 - P(x,a,b)$$

Com essas funções D (decrescente) e V (vale) podemos representar os conceitos atômicos "NÃO ALTO" e "NÃO MEDIANO" como, respectivamente, <não> <alto> e <não> <mediano>.

Para a caracterização de conceitos complexos é utilizado, principalmente, o operador <e>, definido como:

$$A \text{ <e> } B = (a^{*}b)^{**}.5$$

O operador <ou> é definido como:

$$\text{<não> } (A \text{ <ou> } B) = (\text{ <não> } A) \text{ <e> } (\text{ <não> } B)$$

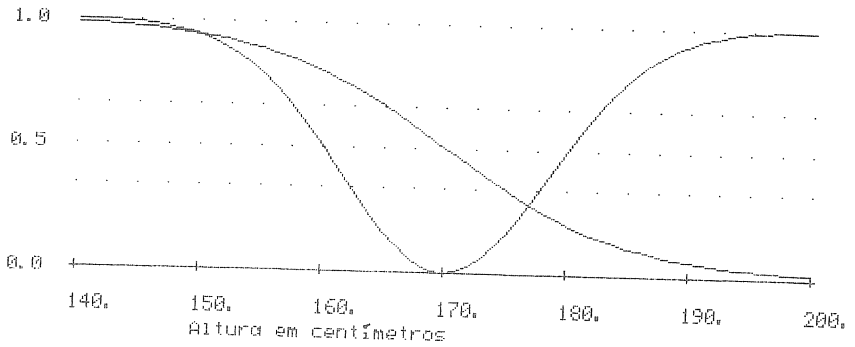


Fig. 2 - ""NÃO ALTO"" e ""NÃO MEDIANO""

Os operadores <e> e <ou> satisfazem:

A <e> B = B <e> A
A <e> A = A
A <e> 0 = 0

A <ou> B = B <ou> A
A <ou> A = A
A <ou> 1 = 1

Com mais de dois operandos, o operador <e> é definido como:

X_1 <e> X_2 <e>... <e> X_n = $(X_1 * X_2 * \ldots * X_n)$ ** (1/n)

Deve-se notar que o operador <e> , como definido, não é associativo, pois expressões de tipo:

A <e> B <e> C
(A <e> B) <e> C
A <e> (B <e> C)

conduzem, em geral, a resultados diferentes, o mesmo ocorrendo, evidentemente, com o operador <ou>. Essa propriedade encontra correspondência na interpretação de expressões linguísticas e possibilita, de forma simples, a representação de tais expressões.

Uma propriedade importante dos operadores definidos é que, se os operandos são restritos aos valores 0 e 1, os resultados obtidos coincidem com os da álgebra booleana.

Como exemplo, se definimos o atributo "peso medido em Kg" e os conceitos atômicos "PESADO" e "NORMAL" como, respectivamente,

<pesado> = C (x,70,10)

<normal> = P (x,70,10)

podemos definir conceitos complexos como:

<grande> = <alto> <e> <pesado>

<gordo> = <pesado> <e> <não> <alto>

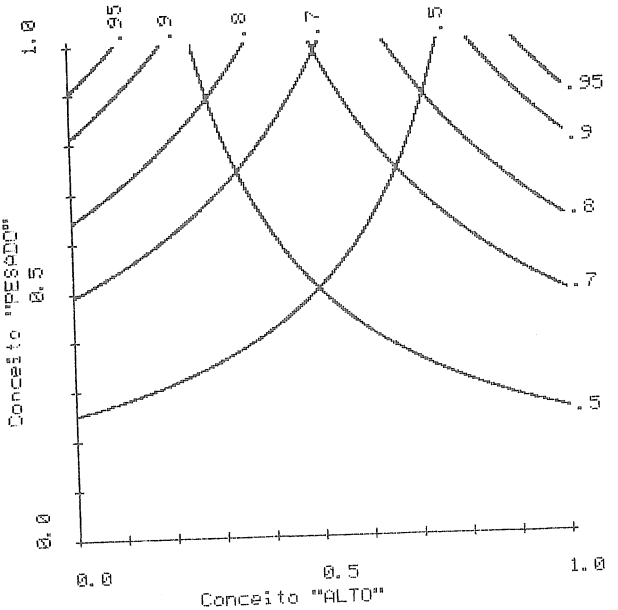


Fig. 3 - Conceitos "GRANDE" e "GORDO"

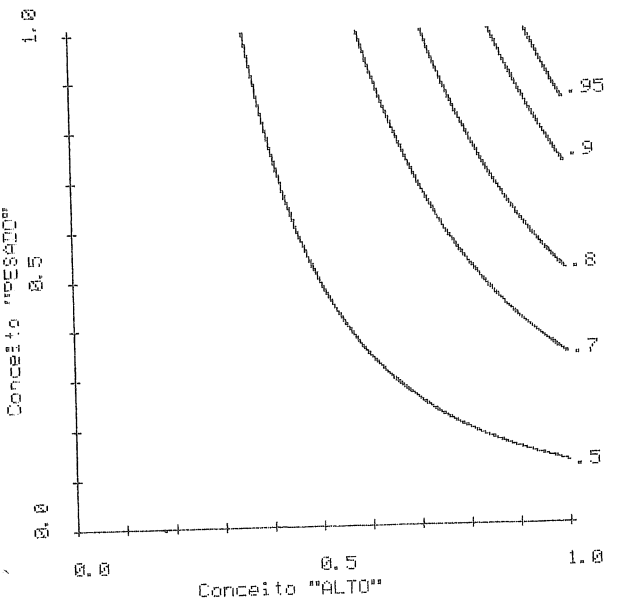


Fig. 4 - ALTO e ALTO e PESADO

Podemos definir conceitos através de expressões mais elaboradas, como:

<grande1> = <alto> <e> <alto> <e> <pesado> ou

<grande2> = <alto> <e> (<alto> <e> <pesado>) =
= <alto> <e> <grande>

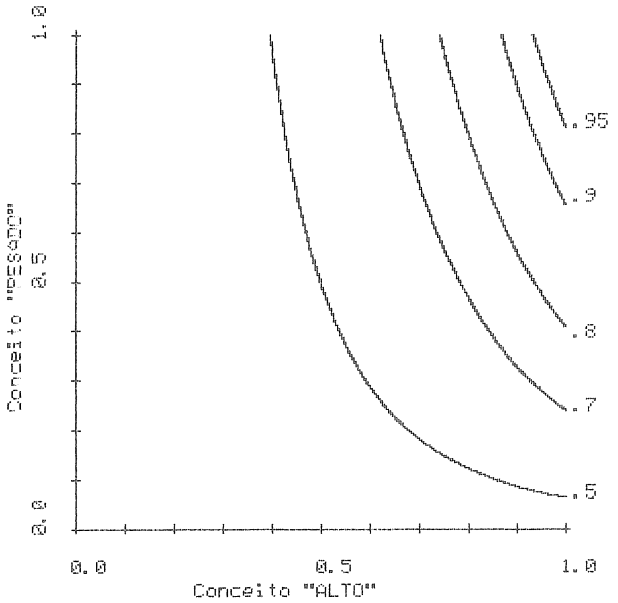


Fig. 5 - ALTO e [ALTO e PESADO]

PROCESSADOR DE SIMILARIDADE

Uma observação é caracterizada por um conjunto de valores correspondentes aos atributos que lhe são aplicáveis. Essa situação define um ponto no espaço conceitual ou em um sub-espaço, pois alguns atributos podem não ser aplicáveis ou observáveis.

O processador de similaridade verifica o grau de inclusão do ponto representativo da observação em cada uma das regiões correspondentes aos conceitos, produzindo como resultado uma lista dos conceitos ordenados segundo aqueles graus de inclusão.

A similaridade de uma observação com um conceito é medida, por definição, pelo próprio valor da função que define o conceito, tomando como valores dos atributos aqueles que caracterizam a observação.

A eficiência do processo depende fortemente da estruturação imposta sobre os conceitos, no sentido de ser evitada a avaliação exaustiva dos índices de relevância do objeto observado com cada um dos conceitos presentes na base de conhecimentos.

IMPLEMENTAÇÃO DO SISTEMA

Um protótipo do sistema em desenvolvimento foi implementado no VAX-11/780 do Instituto de Física da Universidade Estadual de Campinas, como base experimental para a definição detalhada dos requisitos funcionais para o desenvolvimento de uma versão operacional em micro-computador.

O sistema implementado opera sobre quatro arquivos:

ATRIBUTOS: contem os atributos definidos no universo de discurso, suas características e unidades de medida.

CONCEITOS ATÔMICOS: contem os conceitos atômicos e suas funções de pertinência definidas sobre os atributos. Deve ser definido pelo menos um conceito atômico sobre cada atributo.

CONCEITOS COMPLEXOS: contem os conceitos complexos e suas funções de pertinência definidas sobre os outros atributos.

OBSERVAÇÕES: contem a identificação dos objetos, a medida de seus atributos e os índices de pertinência dos conceitos mais similares ao objeto.

Sobre esses arquivos operam quatro procedimentos, que são os módulos de instrução, de observação, de reconhecimento e de explanação.

O módulo de instrução funciona como um editor de conceitos e de atributos, permitindo a inclusão, alteração ou exclusão de atributos, conceitos atômicos e conceitos complexos, garantindo a integridade da estrutura.

O módulo de observação estabelece um diálogo com o operador, visando a identificação do objeto observado e o registro da medida ou avaliação de cada um de seus atributos.

O módulo de reconhecimento avalia os índices de pertinência dos objetos aos conceitos atômicos e complexos, podendo operar com informação incompleta.

O módulo de explanação exibe os conceitos mais pertinentes ao objeto, em ordem decrescente de seus índices, e outras informações complementares relevantes.

CONCLUSÃO

As aplicações potenciais do sistema proposto são variadas e capazes de abranger um amplo universo, pois é função exclusiva do usuário a definição do caráter mais generalista ou especialista do sistema, bem como os resultados obtidos dependem, fundamentalmente, do conteudo da memória de conceitos, que é criada pelo usuário.

O sistema pode ser hierarquizado, de forma a que em um dado ambiente de memória a observação de um objeto conduz ao seu reconhecimento, por hipótese, como um animal; um sistema subordinado (novo ambiente de memória) pode ser invocado, requisitadas observações de novos parâmetros, e o objeto em observação reconhecido como um mamífero. Outro ambiente de memória, mais especializado, e a observação de novos parâmetros, mais específicos, podem conduzir ao reconhecimento do objeto como um cão. A especialização dos ambientes pode continuar no sentido de, por exemplo, reconhecer a raça do cão ou seu estado de saúde.

Um aspecto fundamental do sistema é a sua capacidade potencial de aprendizagem. Se houver acumulação de experiência pelo colecionamento dos reconhecimentos efetuados pelo sistema, enriquecido pela informação prestada pelo usuário do conceito correto ou confirmado relacionado a cada observação, essa informação pode ser utilizada para corrigir e/ou ajustar o banco de conhecimentos atrvés da caracterização dos conceitos.

As idéias e resultados aqui relatados fazem parte de um projeto de pesquisa em desenvolvimento no Departamento de Engenharia Elétrica da Faculdade de Engenharia de Campinas, com apoio do Instituto de Física "Gleb Wataghin" da Universidade Estadual de Campinas.

BIBLIOGRAFIA

Findler, N.V. *Associative networks:* representation and use of knowledge by computers. New York, Academic Press, 1979.

Lindsay, P.H. & Norman, D.A. *Human information systems:* an introduction to psychology, 2. ed. New York, Academic Press, 1977.

Moon, R.E. et alii. *Human-like reasoning capability in a medical diagnostic system:* the application of fuzzy set theory to computerized diagnosis. Working paper 78-001, Canada, University of Toronto/dep. of Industrial Engineering, 1978.

Shortliffe, E.H. & Buchanan, B.G. A model of inexact reasoning in medicine. *Mathematical Biosciencies, 23:135-73, 1975.*

Stefik, M. et alii. The organization of expert systems: a tutorial. *Artificial Intelligence, 18:135-76, 1982.*

Szolovits, P. & Pauker, S.G. Categorical and probabilistic reasoning in medical diagnosis. *Artificial Intelligence, 11:115-44, 1978.*

Zadeh, L.A. *Fuzzy sets and their applications to cognitive and decision processes.* New York, Academic Press, 1975.

Jorge LOBO *

RESUMEN

Este trabajo esboza un formalismo para interpretar un subconjunto de frases en Castellano para interrogar una Base de Datos. El formalismo permite decribir las dependencias que exiten en una frase entre su semantica y su sintaxis, las diferentes jerarquias posibles entre sus componentes y la ambiguedad.

ABSTRACT

This paper sketches a formalism for interpreting a subset of Spanish for querying data bases. This formalism allows us to describe the relationship between semantics and syntax in a sentence, the hierarchy of its various parts and ambiguity.

*Ingeniero en Computacion (USB 83); programacion en Logica, Demostracion automatica, tratamiento de lenguaje natural, Inteligencia Artificial. Departamento de Matematicas y Ciencia de la Computacion, Universidad Simon Bolivar, Apartado 80659, Caracas, Venezuela.

INTRODUCCION

Gran variedad de problemas se presentan cuando se intenta interpretar cualquier frase en lenguaje natural. Sin embargo fuertes restricciones se imponen naturalmente cuando una persona interroga una base de datos. Por ejemplo, el contenido semantico de sus preguntas estara limitado por el contenido de la base de datos.

En esta aplicacion, solo se intenta interpretar frases aisladas cuyo contexo venga determinado por la misma frase y la base de datos (muy probablemente no habran pronombres personales).

Se define un Modelo de Interpretacion, que sirve para darle significado a las preguntas dentro de la Base de Datos. Luego, esta interpretacion es usada para dar las respuestas a dichas preguntas.

Todos los ejemplos presentados estaran basados en el Castellano, sin embargo las ideas basicas pueden ser adaptadas en la interpretacion de otros lenguajes.

EL SUBCONJUNTO DEL LENGUAJE Y EL MODELO DE INTERPRETACION

LOS MUNDOS

Definicion1:
Sea $A=\{a,b,c,\ldots,z\} \cup \{0,1,2,3,\ldots,9\}$, y A^* el monoide libre generado por A, los atomos son los elementos de $A'-\lambda = A+$.

Definicion2:
El conjunto T de terminos de base es el subconjunto mas pequeno de $(A+ \cup \{'(',')'\})+$ tal que:

a) si $a \in A$ entonces $a \in T$
b) si $a \in A$ y $t_1, t_2, \ldots, t_n \in T$ para algun $0<n$ entonces $a(t_1, t_2, \ldots, t_n) \in T$.

Definicion3:

Sean nom, ard y arg funciones definidas como:

nom: T --> A+

i) si $t \in A+$ entonces nom(t) = t
ii) si $t = a(t_1, \ldots, t_n)$ entonces nom(t) = a

ard: T --> N

i) si $t \in A+$ entonces ard(t) = 0
ii) si $t = a(t_1, \ldots, t_n)$ entonces ard(t) = n

arg_t : Ard_t --> T, $Ard_t = \{n/\ 1<n<ard(t),\ t \in (T - A+)\}$

si $t = a(t_1,\ldots,t_n)$ entonces $arg_i(i) = t_i$

Definicion4:

Sea $D=\{D_1,\ldots,D_n\}$ una familia de conjuntos no vacios de terminos en $(T - A+)$. Los conjuntos D_i son llamados dominos de definicion y al conjunto D conjunto de objetos. Una relacion R es un conjunto de terminos que cumplen con las siguientes propiedades:

a) $\forall t_1,t_2 \in R \quad nom(t_1) = nom(t_2)$
b) $\forall t_1,t_2 \in R \quad ard(t_1) = ard(t_2)$
c) $\forall t \in R\ \forall i \in Ard_t\ \exists K \in [1..n] \quad arg_t(i) \in D_k$
d) $\forall t_1,t_2 \in R\ \forall i \in Ard_{t_1}\ \exists K \in [1..n]\ arg_{t_1}(i) \in D_k \Rightarrow arg_{t_2}(i) \in D_k$

Definicion5:

Un mundo M esta definido por la 2-upla $\langle O,R\rangle$ donde:

i) O es un conjunto de objetos
ii) $R = \{R_1,\ldots,R_n\}$ una familia de relaciones definidas sobre O

Cada relacion no vacia R_i en R tiene un nombre asociado el cual esta dado por nom(t) , con $t \in R_i$.

Una Base de Datos es un Mundo cualquiera

EL LENGUAJE Y LAS CATEGORIAS SINTACTICAS

Dentro de una frase se pueden diferenciar grupos de palabras a las que se les asocia un nombre dependiendo del papel que juegan en una oracion. Por ejemplo, en la frase:

El perro mordio al policia

A "perro" y "policia" se les llama nombres comunes, a "El perro" sujeto, "al polica" complemento directo, "El perro mordio al policia" frase. Estos grupos se conocen como categorias sintacticas[STO77], y es sobre ellas que se hace la interpretacion. Notese que al interpretar todas las categorias sintacticas se esta interpretando las frases.

Aunque en general, decidir cuales son las categorias sintacticas es bastante simple, no hay una definicion precisa para ellas. Sin embargo existe un consenso en casi todo lenguaje para caracterizarlas (cosa que no sucede para los casos en las "case-gramar"[FILL68]). En los ejemplos aqui presentes se usaran categorias conocidas del Castellano.

EL SUBCONJUNTO DEL LENGUAJE

Definicion6:

Art = { el, los, la, las, un, una, unos, unas,
algun, alguno, alguna, alguno de los,
alguna de las, algunas, algunos,
todos los, todas las, ningun, ninguna,
ninguna de las, ninguno de los }

Pre = { a, con, de, sobre, en, por, hasta }

Pro_rel = { que, donde, quien, quienes,
el cual, los cuales }

Pro_int = { que, cual, cuales, cuanto, cuantos
cuando, donde, quien quienes }

Ser = { x /x es una conjugacion del verbo ser }

Estar = { x /x es una conjugacion del verbo estar }

Haber = { x /x es una conjugacion del verbo haber }.

Definicion7:

El conjunto VB de vocabulario basico es:

VB = Art U Pre U Pro_rel U Pro_int U Ser U Estar U Haber.

Definicion8:

Los nombres comunes es un subconjunto finito NC de A+

Los adjetivos es un subconjunto finito AD de A+ tal que
$NC \cap AD = \emptyset$

Los nombres propios es un subconjunto finito NP de A+ tal que
$NC \cap NP = \emptyset$, $AD \cap NP = \emptyset$.

Los verbos es un subconjunto finito VE de A+ tal que
$NC \cap VE = \emptyset$, $AD \cap VE = \emptyset$, $NP \cap VE = \emptyset$.

Aunque estos conjuntos estan definidos de manera general, ellos corresponden a las ideas intuitivas de nombres comunes, nombres propios, adjetivos y verbos de los cuales se usaran propiedades conocidas.

Definicion9:

El conjunto VBD del vocabulario de la base de datos es:

$VBD = NC \cup AD \cup NP \cup VE$

Definicion10:
El vocabulario es el conjunto V formado por

$$V = VB \cup VDB$$

La definicion del lenguaje se hace atraves de una garmatica libre del contexto ("context-free")[AHO73] que "exprese" las categorias sintacticas. Una gramatica para algunas frases del castellano podria ser:

```
pregunta --> frase

frase --> "verbo"
      --> sujeto frase
      --> frase modificador
      --> frase comp_directo
      --> frase comp_indirecto
      --> frase mod_nombre

sujeto --> grupo_nominal

comp_directo --> grupo_nominal

comp_indirecto --> modificador

modificador --> "preposicion" grupo_nominal

mod_nombre --> grupo_nominal

grupo_nominal --> nombre
              --> grupo_nominal modificador
nombre --> "articulo" "nombre comun"
       --> "nombre propio"
```

Claramente esta gramatica define algunas frases del Castellano, pero incluye otras frases que no pertenecen. Por ejemplo: frases con dos sujetos o con dos complementos directos, no se verifican el genero y el numero, verbos intransitivos con complemento directo, etc. Estas restricciones o propiedades podran ser verificadas al momento de interpretar la frase y evitar que se reconozcan.

Definicion11:
Sea G = < E, N, P, S > una gramatica libre del contexto [AHO73], sea $c \in E+$, c es una categoria sintactica si:

$S \stackrel{*}{\Longrightarrow} \alpha C \beta$; $\alpha, \beta \in (N \cup E)^*$, $C \in N$ y

$C \stackrel{*}{\Longrightarrow} c$

LA SEMANTICA DE LAS FRASES

Se definira un lengueje de formulas de primer orden; luego, a cada

frase se le asociara una formula de este lenguaje. Por ultimo, se le dara sentido a las frases dandole intrepretacion a las formulas.

LAS FORMULAS DE PRIMER OREDEN

Definicion12:
-Sea V un conjunto infinito numerable de variables.
-Sea C1 = {¬,∨,∧,=>}el conjunto de conectores logicos
-Sea Q = {∀,∃,∃(+),∃(n)} el conjunto de cuantificadores

Definicion13:
El conjunto Tv de terminos es el subconjunto mas pequeno de (A+ U {'('','')'} U V)* tal que:

i) si $v \in V$ entonces $v \in Tv$
ii) si $t \in T$ entonces $t \in Tv$
iii) si $a \in A+$ y $t_1, t_2, \ldots, t_n \in Tv$ para algun $n>=1$ entonces $a(t_1, t_2, \ldots, t_n) \in Tv$.

Definicion14:
Dado un conjunto $P \subseteq A+$, el conjunto F de las formulas es el subconjunto mas pequeno de (P U Tv U CL U Q)+ tal que:

i) si $p \in P$ y $t_1, \ldots, t_n \in Tv$ para algun $n>0$ entonces $p(t_1, \ldots, t_n) \in F$
ii) si $f \in F$ entonces $\neg f \in F$
iii) si $f_1, f_2 \in F$ entonces
$(f_1 \vee f_2) \in F$
$(f_1 \wedge f_2) \in F$
$(f_1 => f_2) \in F$
iv) si $x \in V$, $f \in F$ y $q \in Q$ entonces $qxf \in F$.

Definicion15:
Sea $f \in F$, sea $x \in V$. Una ocurrencia de x en f es cualquier aparicion de la variable x en f.

Definicion16:
Sea g una formula de la forma qxf, con $x \in V$, $q \in Q$ y $f \in F$. f es el alcance del cuantificador q.

Definicion17:
Una ocurrencia de una variable x dentro de una formula f se dice ligada si ocurre dentro del alcance de un cuantificador.

Toda ocurrencia no ligada de una variable x en una formula se llamara libre.

LA INTENSION

Existe una estrecha relacion entre las categorias sintacticas y la gramatica que define al lenguaje. Haciendo uso de esa relacion, se define la intension de cada categoria sintactica por medio de una

extension de la gramatica "context-free" inicial, esta nueva gramatica se fundamenta en las gamaticas de metamorfosis definidas en [COL78]. La intension de una categoria sintactica es un termino en Tv.

Definicion18:
Sea G = < E, N, P, S> una garamatica "context-free", donde sus producciones estan numeradas. Se define una gramatica de intrpretacion como una 2-upla <G',INT>, donde:

i) G' = < E', N', P', S'>
ii) E' = E
iii) Si a ∈ N entonces a(x) ∈ N', con x ∈ V
iv) Si a --> A1 ... An es j-esima produccion en P con a ∈ N, Ai ∈ N U E, 0<i<=n, entonces a(x) --> A1S1 ... AnSn {intensionj(S1',...,Sn',x)} ∈ P'

donde
x ∈ V
Si = λ si Ai ∈ E
Si = (xi) si Ai ∈ N, con xi ∈ V
Si' = Ai si Ai ∈ E
Si' = xi si Ai ∈ N

v) INT ⊆ Tv, llamado intension de los terminos

Ejemplo:
Para la gramatica anterior, las reglas:

(2) frase --> sujeto frase
(10) modificador --> "preposicion" grupo_nominal

se transforman en

frase(x) --> sujeto(x1) frase(x2) {intension2(x1,x2,x)}
modificador(x) --> "preposicion"
grupo_nominal(x2)
{intension10("preposicion",x2,x)}

Definicion19:
Sea t ∈ Tv y sea INT ⊆Tv. t es valido en INT ssi t ∈ INT

Definicion20:
Sea una garamtica de interpretacion <G',INT>, con G' = < E', N', P',S'>; A, B ∈ (E' U N')+

A $\vdash^{0}$ B ssi A = B
A $\vdash^{i+1}$ B ssi u,v,s,t Tv+
A $\vdash^{i}$ uvs, v' --> t' {R} ∈ P', B = uts
y existe sustitucion θ tal que:
vθ = v', tθ = t', y
Rθ es valido en INT.

A $\vdash^{*}$ B ssi $\exists i>=0$ tal que A $\vdash^{i}$ B.

Definicion21:
El lenguaje generado por <G',INT> es

$$L(G',INT) = \{ x \in E'^{+} / S' \vdash^{*} x \}$$

En definitiva, lo que se intenta es asociar a cada categoria sintactica C una relacion S xS ...xS xS donde cada S es la intension de una categoria sintactica que compone a C y S es la intension asociado a C.

Definicion22:
i) el atomo none es una lista
ii) si t $\in$ Tv y L es una lista entonces
lista(t,L) es uan lista.

Por comodidad se usara la notacion de listas definida para PROLOG[CLO81]. (note que las listas pertenecen a Tv).

A continuacion se "define" un subconjunto de INT para la gramatica del ejemplo. Aqui, se mostrara como se pueden resolver algunos problemas de jerarquia en los componentes de las oraciones.

```
pregunta(X) --> frase(X1)
                {intension0(X1,X)}

frase(X) --> "verbo"
             {intension1("verbo",X)}
       --> sujeto(X1) frase(X2)
           {intension2(X1,X2,X)}
       --> frase(X) modificador(X)
           {intension3(X1,X2,X)}
       --> frase(X) comp_directo(X)
           {intension4(X1,X2,X)}
       --> frase(X) comp_indirecto(X)
           {intension5(X1,X2,X)}
       --> frase(X) mod_nombre(X)
           {intension6(X1,X2,X)}

sujeto(X) --> grupo_nominal(X1)
              {intension7(X1,X)}

comp_directo(X) --> grupo_nominal(X1)
                    {intension8(X1,X)}

comp_indirecto(X) --> modificador(X1)
                      {intension9(X1,X)}

modificador(X) --> "preposicion" grupo_nominal(X1)
                   {intension10("preposicion",X1,X)}
```

```
mod_nombre(X) --> grupo_nominal(X1)
                  {intension11(X1,X)}

grupo_nominal(X) --> nombre(X1)
                     {intension12(X1,X)}
                 --> grupo_nominal(X1) modificador(X2)
                     {intension13(X1,X2,X)}
nombre(X) --> "articulo" "nombre comun"
              {intension14("articulo","nombre comun,X)}
        --> "nombre propio"
            {intension15("nombre propio",X)}
```

ALGUNAS DEFINICIONES DE
LAS RELACIONES DE INTENSION

intension0(Fr,For) ∈ INT ssi
Fr = fr(verbo, Complemntos, Lista)
Verificando que la lista posea un sunconjunto minimo de Complementos para que la frase formada con el verbo tenga sentido
For = una formula de primer orden obtenida a partir de Lista

intension1(verbo, Fr) ∈ INT ssi
Fr = fr(verbo,Complementos,[])
Donde complementos es una lista de los posibles sujetos o complementos que puede tener el verbo.

intension2(Suj,Frv,Frn) ∈ INT ssi
Suj = suj(Nc,W,F,Art)
Frv = fr(verbo,Complementos,Lista_vieja)
Frn = fr(verbo,Complementos,Lista_nueva)
Donde Lista_nueva es igual a la lista vieja incluyendo el termino suj(Nc,W,F,Art) si el nombre comun Nc puede ser sujeto del verbo y si el numero del sujeto coincide con el del verbo

intension7(Gn,Suj) ∈ INT ssi
Gn = gn(Nc,W,F,Art,Lista_de_modificadores)
Suj = suj(Nc,W,F1,Art)
Donde F1 es una fromula de primer orden construida a partir la lista de modificadores la formula F, el nombre comun Nc y el artiulo Art.

intension12(Nom,X) ∈ INT ssi
Nom = nom(Nc,W,F,Art)
X = gn(Nc,W,F,Art,[])

intension13(Gn,Mod,X) ∈ INT ssi
Gn = gn(Nc,W,F,Art, Lista_de_modificadores)
Mod = mod(Pp,Nc1,W1,F1,Art1)
Si el nombre comun Nc puedes ser modificado por el nombre comun Nc1 atraves de la preposicion Pp

X = Gn agregando a la lista de modificadores el termino mod(F1,t), donde ocurre W y W1 en t, t ∈ Tv. (Aqui t debe ser una tupla del Mundo)
W, W1 ∈ V

intension14(Art,Nc,Z) ∈ INT ssi
El genero y el numero entre X e Y coinciden,
Z = nom(Nc,W,true,Art)
con W ∈ V

intension15(Np,X) ∈ INT ssi
X = nom(W,Np,true,el)

Es importante notar que la construccion de la formula se puede retardar todo lo que se desee. En este caso, la formula asociada a un grupo nominal no se construye hasta tener todos los modificadores del nombre principal, esto permite hacer cualquier verificacion en el momento de decidir en que cuantificadores se traducen los articulos y cual va a ser su alcance. Todas las reglas propuestas en [PIQ82] e incluso otras mas poderosas pueden ser tratadas[LOB85].

Otro tanto se puede hacer al retardar la construccion de la formula asociada a un verbo. Ademas, definir la intension como relaciones permite darle distintos significados a una palabra[LOB85]. Esto ultimo, mas la ambiguedad de la gramatica permiten dar distintos significados a una frase, cosa comun en cualquier lenguaje natural.

Falta, por ultimo, definir la semantica de las fromulas. Se trabajara con una logica de tres valores para poder soportar las presuposiciones falsas ("El Libro de Pedro es azul" es falso no implica que "El Libro de Pedro no es azul" sea cierto ya que Pedro pude no poseer ningun libro)[PIQ82][KEE72]

La interpretacion de una formula es una funcion que indexada por un mundo da un valor de verdad:

$$I_M: F \longrightarrow \{verdadero, falso, indefinido\}$$

Notacion:
G{v1,...,vn} es una formula tal que su conjunto de variables libres es subconjunto de {v1,...,vn}.

Valores de Verdad:

"verdadero" se notara 1
"falso" se notara 0
"indefinido" se notara i

Notacion:
B = { 1, 0, i}.

Definicion23:
no, and, or, e if son funciones definidas como:

no: B --> B
and: BxB --> B
or: BxB --> B
if: BxB --> B

no[1] = 0, no[0] = 1, no[i] = i,
$\forall v \in B$
and[1,0] = and[0,1] = 0,
and[i,v] = and[v,i] = i,
and[1,1] = 1,
or[1,0] = or[0,1] = or[1,1] = 1,
or[i,v] = or[v,i] = i,
or[0,0] = 0,
if[0,v] = if[i,v] = i,
if[1,v] = v.

Definicion24:
Sea $X \subseteq V$, $\mathcal{O} \subseteq T$, $g \in F$ y $\Theta_{x,\mathcal{O}}$una funcion cualquiera de X en O.
Sea {v1,...,vn} $\in$ X, O<n.

g[Θ/{v1,...,vn}] es la fromula que resulta de sustituir toda ocurrencia libre de vi en g por $\Theta_{x,\mathcal{O}}$(vi). $\Theta_{x,\mathcal{O}}$es una sustitucion.

Definicion25:
Dado g $\in$ F y un mundo M = <O,R>, siendo O = {D_1,...,D_n}.
Sea O = $\bigcup_{1 \le i \le n}$ Di

i) Si g{v1,...,vn} es de la forma p(t_1,...,t_n), X = {v1,...,vn}, con p $\in$ A+ y ti $\in$ Tv entonces
si existe $\Theta_{x,\mathcal{O}}$ y una tupla t en R tal que
g[Θ/X] = t
entonces I_M(g) = 1
sino I_M(g) = 0
ii) Si g = $\neg$f, f $\in$ F
entonces I_M(g) = no[I_M(f)]
iii) Si g = f $\wedge$ f'; f,f' $\in$ F entonces
I_M(g) = and[I_M(f),I_M(f')]
iv) Si g = f $\vee$ f'; f,f' $\in$ F entonces
I_M(g) = or[I_M(f),I_M(f')]
v) Si g = f $\Rightarrow$ f'; f,f' $\in$ F entonces
I_M(g) = if[I_M(f),I_M(f')]
vi) Si g = $\forall$xf, f $\in$ F entonces
si existe $\Theta_{x,\mathcal{O}}$ tal que I_M(f[Θ/{x}]) = 0
entonces I_M(g) = 0
sino
si existe $\Theta_{x,\mathcal{O}}$ tal que I_M(f[Θ/{x}]) = 1
entonces I_M(g) = 1
sino I_M(g) = i
vii) Si g = $\exists$xf, f $\in$ F entonces
si existe $\Theta_{x,\mathcal{O}}$ tal que I_M(f[Θ/{x}]) = 1
entonces I_M(g) = 1

sino
si existe $\Theta_{[x],\sigma}$ tal que $I_{IM}(f[\Theta/\{x\}]) = 0$
entonces $I_{IM}(g) = 0$
sino $I_{IM}(g) = i$

viii) Si $g = \exists(+)f$ entonces
si existen $\Theta_{[x],\sigma} \neq \Theta'_{[x],\sigma}$ tales que
$I_{IM}(f[\Theta/\{x\}]) = I_{IM}(f[\Theta'/\{x\}]) = 1$
entonces $I_{IM}(g) = 1$
sino
si existe $\Theta_{[x],\sigma}$ tal que $I_{IM}(f[\Theta/\{x\}]) \neq i$
entonces $I_{IM}(g) = 0$
sino $I_{IM}(g) = i$

ix) Si $g = \exists(n)f$, $n \in N$ entonces
si existen $\Theta^1_{[x],\sigma}, \ldots, \Theta^n_{[x],\sigma}$ sustituciones distintas
tales que $I_{IM}(f[\Theta^i/\{x\}]) = 1$
entonces $I_{IM}(g) = 1$
sino
si existe $\Theta_{[x],\sigma}$ tal que $I_{IM}(f[\Theta/\{x\}]) \neq i$
entonces $I_{IM}(g) = 0$
sino $I_{IM}(g) = i$.

Un modelo de interpretacion M es una 4-upla <O,R,G,INT> donde

<O,R> es un mundo
<G,INT> una gramatica de interpretacion.

CONCLUSIONES Y COMENTARIOS

Un desarrollo completo de las gramaticas de interpretacion estan en [LOB85]. Alli se presenta un ejemplo para un conjunto mas amplio del Castellano. Para definir la relacion de intension y la construccion de las formulas se siguieron las ides presentes en [BAT84][McC82][DIK78][PER82]. Ademas, se esta desarrollando un sistema en PROLOG, como prototipo para probar las posibilidades y limitaciones del formalismo.

El retardo de las deciciones durante el analisis permite retardar el manejo de los errores hasta poseer la mayor informacion posible. Un manejo de errores, necesario en este tipo de aplicaciones debido a la imposibilidad de abarcar todas las frases del lnguaje natural (problema que se ha tratado poco hasta ahora) puede facilitarse con este enfoque.

AGRADECIMIENTOS:

Agradesco la colaboracion prestada por mi tutor Gerard Battani, quien ha guiado la tesis en que se inspira este trabajo, y a Maria Antonia Mozota por su interes y criticas sobre estas lineas.

BIBLIOGRAFIA

[BAT83] Gerard BATTANI
"Formalizacion de Lenguas Naturales en Logica e Interrogacion de Bases de Datos en Castellano", Reporte No. 83-21 Dpto. de Matamaticas y Ciencia de la Computacion. Universidad Simon Bolivar (1983)

[CLO81] W. F. CLOCKSIN, C. S. MELLISH
"Programming in Prolog" Springer-Verlag (1981)

[COL78] Alain COLMERAUER
"Metamorphosis grammars"
in Lecture Notes in Computer Science No 63, Springer Verlag (1978)

[DIK78] Simon C. DIK
"Functional Grammar" North-Holland Linguistic Series (1978)

[KEE72] E. L. KEENAN
"On Semantically Based Grammar", Linguistic Inquiry vol.III,4 (Fall 1972) 413-459

[LOB85] J. Jorge LOBO
"Prototipo de un Sistema de Interrogacion de Bases de Datos en Castellano", Tesis de Maestria en desarrollo, Universidad Simon Bolivar
Caracas (1985)

[McC82] Michael C. McCORD
"Using Slots and Modifiers in Logic Grammars for Natural Language" Artificial Intelligence vol.18 (1982) pp. 327-367

[PER83] Fernando PEREIRA
"Logic for Natural Language Analysis" SRI International, Technical Note 275, (January 1983)

[PIQ82] Jean Francois PIQUE
"On a Semantic Representation of Natural Language Sentences" in Proceedings of The First International Logic Programming Conference, Marseille (1982) pp.215-223.

[STO77] Robert P. STOCKWELL
"Foundations of Syntactic Theory" Prentice-Hall (1977)

S. Lemos Meira◇

Sumário

Alguns aspectos de complexidade de algoritmos puramente funcionais são considerados, dentro de um modelo de execução de linguagens de alta ordem, não estritas. O problema de espaço residente criado pelo critério de execução é estudado para alguns exemplos, e soluções são propostas para os casos analisados, usando uma variação da técnica de programação transformacional.

Abstract

We consider some aspects of complexity of purely applicative algorithms, in higher-order, non-strict languages. We study the increased residency problem (if compared to the corresponding imperative versions of the same algorithms) due to lazy evaluation of the language, which is exemplified in some basic definitions. For the cases in question, solutions are proposed using a variation of the methods of transformational programming.

◇ Engenheiro de Eletrônica (ITA, 1977), M. Sc. em Computação (UFPE, 1981), Ph. D. in Computing (Kent, UK, 1985). Áreas de interesse: linguagens funcionais, complexidade de algoritmos, teoria da computação, arquiteturas não-von-Neumann. Endereço: Departamento de Eletrônica e Sistemas, UFPE, Cidade Universitária, 50.000, Recife-PE, Brasil.

1. Introdução: Porque Linguagens Funcionais?

Nesta Introdução, o nosso objetivo é mostrar porque as linguagens funcionais são necessárias à atividade de produção de software, visto como ciência ou engenharia.

Há muito tempo que se fala na *crise de software*. Esta, para a computação, pode ser comparada à crise do petróleo, na década de 70, em relação aos seus custos e efeitos na sociedade.

O custo decrescente de hardware, juntamente com o aumento de complexidade dos pacotes de software, e consequente aumento dos custos de especificação, escrita e teste dos mesmos, faz com que se estime, hoje em dia, uma razão de 4 para 1 entre os custos de software e hardware em sistemas integrados [Boehm 81].

Claramente, a ciência de software deveria buscar uma solução para tal problema, da mesma forma que VLSI é uma solução, nem que seja temporária, para o custo de hardware. Analisando o desenvolvimento das linguagens de programação de uso geral, dos anos 50-60 até agora, vê-se que estas não mudaram muito, e em muitos casos, mudaram para pior.

Linguagens de programação deveriam ser usadas para especificar, de forma executável, o problema que o programador quer resolver. Isto significa que uma boa linguagem de programação tem que ser *programmer-friendly*: afinal de contas, ela teria sido criada para facilitar a vida do programador.

Agora consideremos o caso de ADA^{TM} [Honeywell 80]. ADA foi desenvolvida quando já se sabia das causas e efeitos da *crise de software*, e um dos seus objetivos é justamente contornar as causas da *crise*. No entanto, numa análise de complexidade da linguagem, Bjørner e Oest (em [Bjørner et al 80]) dão um grau de complexidade 8 para ADA, relativo a 1 para ALGOL60 e 5 para PL/I. Como os programadores que já tentaram escrever um programa em PL/I devem se lembrar, ela não é exatamente uma linguagem que se possa chamar de fácil, e isto coloca ADA numa posição bastante incômoda.

Aumentando a complexidade da linguagem, aumenta-se a dificuldade de uso, juntamente com a probabilidade de erros de programação, e, talvez muito pior, torna-se impossível a manipulação formal de programas –como transformação e prova de correção. O resultado óbvio é um aumento, *e não diminuição*, do custo de software. Uma possível conclusão é que a mais provável contribuição de linguagens como ADA é agravar, e não resolver, ou mesmo amenizar, a *crise de software*.

Ao invés de linguagens cada vez mais complicadas, as quais, à parte das críticas acima, também não ajudam a produtividade do programador, o que se precisa é de linguagens **simples**, mas que englobem conceitos (de programação) mais poderosos. Este é o caso do estilo tratado neste artigo. A seguir, apresentaremos alguns aspectos da linguagem de programação KRC (Kent Recursive Calculator), desenvolvida na University of Kent at Canterbury, UK, pelo Prof. David Turner. A apresentação é informal e segue a mesma linha de [Turner 82b]. A sintaxe e semântica denotacional formais da linguagem estão especificadas em [Meira 85t].

2. Programação Denotacional[1]

O termo programação denotacional é usado para acentuar o fato de que a nossa preocupação é com a real forma dos objetos dos quais as nossas definições tratam, e não com o processo através dos quais eles são construídos. Um exemplo disso é a definição, em KRC, da função *fatorial*:

$$fatorial\ 0 = 1$$
$$fatorial\ n = n * fatorial\ (n-1).$$

A primeira equação define o caso base para a função, ou seja, o *fatorial* de *0* é *1*. A segunda equação define os casos restantes, recursivamente, e é claro que *fatorial* é uma função total em $\mathbb{N}$, o domínio dos números naturais. Note o uso de *pattern matching* na primeira equação, funcionando como um condicional.

O estilo informal de programação que discutimos aqui é baseado em equações recursivas de alta ordem (≥ 1), não estrito, e usa conceitos da teoria dos conjuntos de Zermelo-Fraenkel [Mendelson 79].

[1] Termo devido a Christopher Strachey.

Neste contexto, não estrito significa que os parâmetros reais de definições, semanticamente, são passados por nome, ou que o mecanismo de execução da linguagem é "preguiçoso" ou *lazy*. Não há atribuição (*assignment*) de valores a variáveis –porque não há conceito de armazenamento, efeitos colaterais não existem, e também não há conceito de controle como em linguagens imperativas.

Em KRC, funções são *cidadãos de primeira classe*, i.e., podem tanto ser argumento como resultado de outras funções. Todas as funções são tratadas como funções de apenas 1 parâmetro. Por exemplo,

$$some\ a\ b = a + b$$

é uma função de um parâmetro que, quando aplicada somente ao parâmetro real *77*, em *some 77*, retorna a função (de 1 parâmetro)

$$(some\ 77)\ b = 77 + b.$$

Tal dispositivo é chamado *currying*[2], e é de fundamental importância para o estilo de programação funcional. KRC é uma linguagem "estática": os valores de objetos descritos na linguagem não mudam com o tempo.

À parte de objetos escalares como números, *strings*, funções e *booleans*, KRC lida com um único tipo estruturado, a **lista**. Listas são coleções de objetos de qualquer tipo –KRC é *weakly-typed*, ver [Strachey 72]– e exemplos básicos de listas são:

$$["semish", 1985] \qquad [fatorial, 12, \mathbf{true}] \qquad [\,],$$

o último dos quais é a lista vazia ou **nil**. Algumas operações definidas sobre listas são:

$a : x$ adiciona o objeto a à frente da lista x, é equivalente à operação (**cons** a x) em LISP;

$hd\ x$ retorna o primeiro objeto da lista x, isto é, o *head* de x;

$tl\ x$ retorna a lista x a menos do seu primeiro objeto, i.e., *tail* de x.

O construtor ":" também pode ser usado em definições baseadas em "pattern matching", de tal forma que podemos definir

$$hd\ (a : x) = a$$
$$tl\ (a : x) = x,$$

e tanto *hd* como *tl* da lista vazia são indefinidos, ou, semanticamente, $\perp$ (*bottom*).

Usando tais conceitos básicos, a função $\sum$ (somatória) sobre uma lista de números pode ser definida como:

$$somat\ [\,] = 0$$
$$somat\ (a : x) = a + somat\ x.$$

A explicação desta definição é trivial. Se assumirmos que **num** representa os objetos de tipo *número* e [*] representa o tipo *lista de* *, o tipo da definição acima é

$$somat :: [\mathbf{num}] \mathbf{->} \mathbf{num},$$

onde "::" é lido *tem tipo*.

Um exemplo de uso de funções de alta ordem é a definição de *map*

$$map\ f\ [\,] = [\,]$$
$$map\ f\ (a : x) = f\ a : map\ f\ x.$$

map é uma função de tipo

$$(* \mathbf{->} **) \mathbf{->} [*] \mathbf{->} [**].$$

[2]Em homenagem a Haskell B. Curry.

O resultado da aplicação *map g l* é aplicar a função *g* a todos os elementos da lista *l*. Para o tipo de dados [[**num**]], ou seja "lista de lista de **num**" (representando matrizes...), pode-se definir a função

$$matsom = map\ somat,$$

que computa a somatória das linhas (poderia ser colunas) da "matriz" à qual é aplicada.

Outra definição que faz uso do mesmo conceito é

$$\begin{aligned} filter\ cond\ [\,] &= [\,] \\ filter\ cond\ (a : x) &= a : filter\ cond\ x, \quad cond\ a \\ &= filter\ cond\ x, \end{aligned}$$

que também serve para demonstrar o uso de equações condicionais (*guarded equations*) em KRC. O resultado de *filter* é a lista de entrada **menos** os elementos para os quais o teste *cond a* é **false**. Por exemplo, para

$$impar\ b = b\ \%\ 2\ \backslash =\ 0,$$

onde $a\ \%\ b$ indica o resto da divisão de a por b e $\backslash =$ representa *diferente de*, podemos definir

$$impares = filter\ impar,$$

a função que elimina números pares da sua lista-argumento.

Note como a junção de *map* e *somat* para formar *matsom* e de *filter* e *impar* para formar *impares* é extremamente simples e natural. Esta modularização e acoplamento, que é possível no tipo de linguagem do qual KRC é um representante, é peça fundamental do estilo de programação que defendemos. Em linguagens imperativas como FORTRAN ou ADA, o exercício acima não pode ser resolvido com a mesma facilidade, e certamente necessitará reescritura do programa inicialmente desenvolvido para implementar *filter* ou *map*. Devido à ausência do conceito de funções de alta ordem, pode-se dizer que *filter* e *map* não *podem ser implementados* em tais linguagens.

Mais um exemplo: *Sorting*

Considere a definição de ordenação por inserção (*insertion sorting*). Esta pode ser implementada em dois passos. Primeiro, define-se uma função que insere um objeto (comparável sob $\leq$) em uma lista já ordenada (sob $\leq$):

$$\begin{aligned} insert\ a\ [\,] &= [a] \\ insert\ a\ (b : x) &= a : b : x, \quad a <= b \\ &= b : insert\ a\ x. \end{aligned}$$

Em segundo lugar, definimos a função que cuida da ordenação

$$\begin{aligned} insort\ [\,] &= [\,] \\ insort\ (a : x) &= insert\ a\ (sort\ x). \end{aligned}$$

Dignos de nota, nesta definição, são a clareza com que o algoritmo foi especificado e a facilidade com que se pode derivar uma prova de sua correção total (ver [Meira 82] para tal). É importante notar também que a complexidade, no tempo, desta definição é a mesma do programa imperativo que implementa o mesmo algoritmo usando *arrays* (ver [Sedgewick 83] por exemplo). A complexidade espacial da definição funcional acima é discutida na Seção 6.

3. Expressões de Zermelo-Fraenkel

Estas formam uma das partes mais interessantes da linguagem KRC. Definições usando expressões ZF são extremamente compactas e auto-explicativas, às vezes ao ponto de serem, ao mesmo tempo, a

especificação, solução e documentação de um problema. Tome como primeiro exemplo a definição ZF da função *impares* já vista anteriormente:

$$impares\ li = \{n \mid n\ <-\ li;\ impar\ n\}.$$

Esta definição é lida "a lista de todos os objetos n tirados da lista li tal que *impar* n".

Um exemplo mais complexo é a definição da função que gera todas as permutações dos elementos de uma lista li,

$$perms\ [\,] = [[\,]]$$
$$perms\ li = \{a : p \mid a\ <-\ li;\ p\ <-\ perms\ (li\ --\ [a])\},$$

onde $--$ é o operador *diferença* sobre listas (e.g. $[1,2,1,3]\ --\ [1] = [2,1,3]$). *perms*, juntamente com um predicado que decide a ordenação de listas (sob $\leq$)

$$orden\ li = and\ \{li\ i\ <=\ li\ (i+1) \mid i\ <-\ [1..\#li - 1]\},$$

onde

$li\ i$ é o i-ésimo elemento da lista li;

$[n..m]$ é a lista $[n, n+1, \cdots, m]$;

$\#li$ é o tamanho da lista li e

$$and\ [\,] = \mathbf{true}$$
$$and\ (\mathbf{true} : x) = and\ x$$
$$and\ (\mathbf{false} : x) = \mathbf{false}$$

pode ser usada para especificar o problema de ordenação sem usar nenhum "algoritmo" específico:

$$sort\ li = hd\ \{p \mid p\ <-\ perms\ li;\ orden\ p\}.$$

Apesar de executável, esta definição tem complexidade fatorial, o que a torna absolutamente inútil na prática.

Para um algoritmo bem mais rápido ($\mathbf{O}(n \log n)$), que encerra os exemplos e a apresentação do estilo de programação, considere nossa definição do algoritmo de *quick sort* [Hoare 62], usando expressões ZF:

$$quick\ [\,] = [\,]$$
$$quick\ (a : x) = quick\ \{b \mid b\ <-\ x;\ b\ <=\ a\}\ ++\ a : quick\ \{b \mid b\ <-\ x;\ b > a\}.$$

A beleza e concisão desta definição é melhor apreciada quando em comparação com definições imperativas do mesmo algoritmo (ver [Wirth 76], pp. 76).

4. Complexidade de Algoritmos Funcionais

A seguir nós consideramos a complexidade das definições funcionais de *fatorial*, *insertion* e *quick sort*, comparando à complexidade dos equivalentes imperativos.

O modelo de execução que adotamos para a análise de complexidade é o de sistemas "preguiçosos" de reescritura (*lazy term rewrite systems* [Huet & Oppen 80]), que é ótimo em relação aos problemas aqui discutidos. Modelos práticos de implementação de linguagens funcionais, tais como *combinadores* [Turner 79a] terão eficiência necessariamente mais baixa, pelo menos por um fator constante (ver [Meira 85t]). Dito isto, a análise aqui apresentada se refere ao limite inferior da complexidade média das definições.

A palavra preguiçoso acima se refere ao fato de que os termos em questão são reescritos em uma ordem *mais-à-esquerda/mais-externa*, e também (implicitamente) que qualquer sub-termo compartilhado é computado no máximo uma vez. Este mecanismo informal é equivalente ao *fully lazy normal graph reduction* para o λ-cálculo [Wadsworth 71] e lógica de combinadores [Turner 81a]. Agora passemos à análise prometida, onde $\mathbf{T}$ é a complexidade temporal e $\mathbf{R}$ é a complexidade espacial em $\mathrm{P_reg}$, o nosso modelo de execução.

5. Fatorial

A nossa definição funcional de *fatorial* é linear no tempo –$\mathbb{T}$ (*fatorial* n) = $\mathbb{O}(n)$– e no espaço –$\mathbb{R}$ (*fatorial* n) = $\mathbb{O}(n)$. O último fato se deve à mesma razão da definição imperativa recursiva ser linear em espaço. Basicamente, para se calcular *fatorial* n, é necessário trabalhar ao contrário e descobrir primeiro o fatorial de 0, 1, etc. Assim sendo, tem-se

$$fatorial\ n = \underbrace{n * ((n-1) * (\cdots * 1))}_{n+1\ sub-termos}$$

o que implica em $\mathbb{R}(fatorial\ n) = \mathbb{O}(n)$. A história, no entanto não acaba aqui.

Como todo principiante sabe, uma alternativa mais eficiente do que o programa imperativo recursivo é um programa *for*,

$$f := 1;$$
$$\textbf{for } c := 1 \textbf{ to } n \textbf{ do } f := f * c;$$

para o qual a complexidade espacial é constante.

Uma alternativa imediata para melhorar a eficiência da definição funcional é reescrevê-la imitando o programa acima, o que é possível usando *tail recursion*:

$$fatorial\ n = tfat\ n\ 1$$
$$tfat\ 0\ res = res$$
$$tfat\ n\ res = tfat\ (n-1)\ (n * res).$$

Infelizmente, a nova definição tem a mesma complexidade da definição puramente recursiva apresentada anteriormente. Em KRC, como na maioria das linguagens não estritas, os parâmetros reais das definições são passados "por necessidade" (*call-by-need*), isto é, com a mesma semântica de *call-by-name*, mas computados uma única vez, quando seu valor é requisitado pelo mecanismo de execução. O segundo parâmetro de *tfat* só tem seu valor requerido quando do término (e consequente impressão) do cálculo de *tfat*. Daí, o cálculo procede segundo os estágios a seguir:

$$fatorial\ n = tfat\ n\ 1 = tfat\ (n-1)\ (n * 1) =$$
$$= tfat\ (n-2)\ ((n-1) * (n * 1)) = \cdots = tfat\ 0\ \underbrace{(1 * \cdots ((n-1) * (n * 1)))}_{n\ termos}$$

com o segundo parâmetro crescendo em tamanho a cada nova chamada recursiva à *tfat*. O primeiro parâmetro ocupa espaço constante, porque é computado a cada chamada, para decidir a terminação de *tfat* (em *tfat 0 res* = *res*).

Na realidade, este comportamento só será eliminado se conseguirmos forçar a computação do segundo parâmetro de *tfat* a cada chamada. Tal procedimento é seguro, pois preserva a semântica da definição, dado que *tfat n res* $\neq \perp$ se e somente se tanto n como *res* são diferentes de $\perp$. Assim, pode-se escrever

$$tfat\ n\ res = res, \quad res > 0\ \&\ n = 0$$
$$= tfat\ (n-1)\ (n * res).$$

Analisando a condição na primeira equação, tem-se

$$res > 0\ \&\ n = 0 \equiv \textbf{true}\ \&\ n = 0 \equiv n = 0,$$

supondo que o domínio de *fatorial* é $\mathbb{N}$. A semântica de &,

$$\perp\ \&\ exp = \perp$$
$$\textbf{true}\ \&\ exp = exp$$
$$\textbf{false}\ \&\ exp = \textbf{false},$$

garante que o segundo parâmetro de *tfat* é sempre computado (porque ele é o primeiro de & e este é sempre computado). Consequentemente, $\mathbb{R}$ (*fatorial* n) = $\mathbb{k}$.

Uma crítica séria a este tipo de procedimento é que ele vai contra os princípios de programação denotacional advogados neste artigo. Em outras palavras, o desenvolvimento anterior mostrou que o programador tem que se preocupar com o método de construção dos objetos com que trata, de forma a tornar sua computação mais eficiente. Enquanto reconhecemos que este possa ser um dos calcanhares de Aquiles deste estilo de programação, a conversa não acaba aqui, e a contra-argumentação tem três aspectos diferentes:

1- Justamente porque a primeira aproximação à solução foi denotacional, é possível aplicar com mais facilidade transformações que tornam a *especificação* mais eficiente, e provar que tais transformações não modificaram o significado da definição por onde começamos;

2- Existem métodos automáticos que podem lidar com, por exemplo, transformação de *call-by-need* para *call-by-value* [Meira 84, Mycroft 81]. Um destes métodos, aplicado a *tfat*, tornará aquela definição tão eficiente quanto a que conseguimos adicionando o teste redundante no condicional;

3- Em linguagens nas quais o programador pode definir tipos abstratos de dados e seus construtores, com a facilidade especial de especificar *leis* que guiam o uso de construtores, como MIRANDA [Turner 84] o trabalho de transformação de programa pode fazer parte da fase de especificação do tipo de dados com que se lida. Por exemplo, se o operador $*$ de um tipo de dado **inteiro** é definido como sendo associativo, a primeira definição de *fatorial* que usamos tem residência constante. Para outros exemplos usando a mesma idéia, ver [Meira 85e].

6. Insertion Sorting

O leitor atento já deve ter descoberto que a definição de *insertion sorting* na Seção 2 tem complexidade espacial linear. Para ordenar uma lista $[l_1, l_2, \cdots, l_n]$ usando *insort*, o mecanismo de execução primeiro cria o fechamento

$$insort\ [l_1, l_2, \cdots, l_n] = insert\ l_1\ (insert\ l_2\ (\cdots (insert\ l_n\ [\,]))),$$

que ocupa espaço $\mathbf{O}(n)$. Como a lista resultante da ordenação tem tamanho n, o uso de espaço por *insort* é constante, $\mathbf{O}(n)$.

O algoritmo imperativo, usando *arrays*, tem complexidade espacial constante (ver [Sedgewick 83]). Para ter complexidade constante, o algoritmo funcional precisa ser modificado de tal forma que o fechamento discutido acima não é criado. Novamente, usando *tail recursion*, definimos

$$\begin{aligned} trinsort\ [\,] &= [\,] \\ trinsort\ (a : x) &= trins\ [\,]\ [a]\ x, \end{aligned}$$

onde *trins* é uma função de tipo

$$trins :: lo -> lo -> [*] -> lo,$$

e *lo* representa o tipo "lista ordenada de $*$". Uma possível definição de *trins* é

$$\begin{aligned} trins\ l\ [\,]\ (b : y) &= trins\ [\,]\ (l\ +\!+\ [b])\ y \\ trins\ l\ x\ [\,] &= l\ +\!+\ x \\ trins\ l\ (a : x)\ (b : y) &= trins\ (l\ +\!+\ [a])\ x\ (b : y), \quad a < b \\ &= trins\ [\,]\ (l\ +\!+\ b : a : x)\ y. \end{aligned}$$

A prova de correção desta definição por indução estrutural [Burstall 69, Meira 82] é imediata.

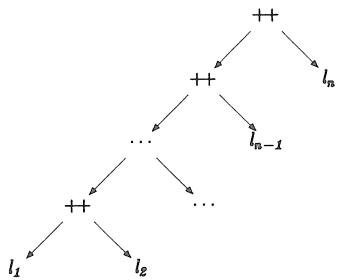
Assumindo que o espaço usado por $[a]\ +\!+\ b$ é o mesmo usado por $a : b$ (obviamente, se b é uma lista, $[a]\ +\!+\ b = a : b$), esta definição usa espaço constante, porque em qualquer estágio da computação o

espaço ocupado por $l \mathbin{+\!\!+} a : x \mathbin{+\!\!+} b : y$ é o mesmo ocupado pela lista argumento de *trinsort*. Resolvido o problema de complexidade espacial, mas infelizmente criamos um problema no tempo, devido ao uso do operador $+\!\!+$ na terceira e quarta equações de *trins*.

Considere a computação de

$$((l_1 \mathbin{+\!\!+} l_2) \mathbin{+\!\!+} \cdots) \mathbin{+\!\!+} l_n,$$

um fechamento da mesma forma que o criado em *trins*. Este é representado pela árvore (degenerada à esquerda)



Por sua vez, o operador $+\!\!+$ (*append*, em forma prefixa) é especificado pela definição

$$\begin{aligned} append\ [\,]\ y &= y \\ append\ (a : x)\ y &= a : append\ x\ y, \end{aligned}$$

donde se conclui que a lista à esquerda de $+\!\!+$ tem que ser percorrida até o último elemento, para que se compute $l_i \mathbin{+\!\!+} l_j$. Na árvore acima, as seguintes listas são percorridas *in totum*:

$$l_1, \quad l_1 \mathbin{+\!\!+} l_2, \quad l_1 \mathbin{+\!\!+} l_2 \mathbin{+\!\!+} l_3, \quad \cdots \quad l_1 \mathbin{+\!\!+} l_2 \mathbin{+\!\!+} l_3 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} l_{n-1}$$

para que se compute a lista final. Isso implica numa complexidade temporal $\mathbb{O}(n^2)$ para tal computação. Em *trins*, um fechamento deste tipo tem que ser computado a cada passo. Em n passos, teremos uma complexidade temporal $\mathbb{O}(n^3)$, o que faz o algoritmo uma ordem de magnitude mais lento que o seu equivalente imperativo.

Para resolver tal problema, considere primeiro a definição

$$\begin{aligned} reverse\ [\,] &= [\,] \\ reverse\ (a : x) &= reverse\ x \mathbin{+\!\!+} [a], \end{aligned}$$

que especifica a função que inverte uma lista, i.e., $l'_1 = l_n$, $l'_2 = l_{n-1}$, etc. A complexidade temporal de *reverse* é $\mathbb{O}(n^2)$, pela mesma razão acima. Mas se a reescrevermos como

$$\begin{aligned} reverse\ x &= rev\ x\ [\,] \\ rev\ [\,]\ y &= y \\ rev\ (a : x)\ y &= rev\ x\ (a : y), \end{aligned}$$

sua complexidade temporal é linear em $\#x$. Aplicando a mesma idéia à definição de *trins*, e usando *reverse* para inverter a lista criada em cada ciclo por *trins* (porque $reverse\ (reverse\ x) = x$), tem-se

$$\begin{aligned} trins\ l\ [\,]\ (b : y) &= trins\ [\,]\ (reverse\ l \mathbin{+\!\!+} [b])\ y \\ trins\ l\ x\ [\,] &= reverse\ l \mathbin{+\!\!+} x \\ trins\ l\ (a : x)\ (b : y) &= trins\ (a : l)\ x\ (b : y), \quad a < b \\ &= trins\ [\,]\ (reverse\ l \mathbin{+\!\!+} b : a : x)\ y. \end{aligned}$$

A complexidade temporal de *trins* agora é $\mathbb{O}(n^2)$, e a espacial é constante, ou seja, nossa definição tem a mesma eficiência do algoritmo imperativo. O leitor pode verificar que as tranformações aplicadas preservam a semântica de *trins*.

O último caso a ser analisado é a definição de *quick sort*. Em implementações imperativas, pode-se ter uma versão deste algoritmo de tal forma que a complexidade temporal média é $\mathbb{O}(n \log n)$, com pior caso $\mathbb{O}(n^2)$, e o pior caso para a complexidade espacial é $\mathbb{O}(\log n)$, quando se elimina recursão. Relembrando nossa definição de *quick sort*

$$\begin{aligned} quick\ [\,] &= [\,] \\ quick\ (a : x) &= quick\ \{b \mid b\ <-\ x;\ b\ <=\ a\}\ ++\ a : quick\ \{b \mid b\ <-\ x;\ b > a\}, \end{aligned}$$

observa-se, em primeiro lugar, que ela não faz nenhuma concessão à eficiência. Por exemplo, a partição das listas é feita simplesmente percorrendo cada lista duas vezes para extrair os elementos $\leq a$ e $> a$. Como já foi visto em casos anteriores, também há uma componente quadrática na complexidade – agora menos séria– devido à complexidade de computação de séries de $++$'s.

Considerando que o problema gerado por $++$ pode ser resolvido como nos casos anteriores, ou usando outras regras de computação para $++$ [Sleep & Holmstrom 82], vamos concentrar a análise no que nos parece o principal problema desta definição, que é o uso de espaço. Primeiro considere o melhor caso, em termos de complexidade espacial, que é a lista de entrada já ordenada (este é o pior caso no tempo!). Para simplificar a análise, assume-se que todos os elementos da lista são diferentes entre si. Neste caso, para a lista ordenada, todas as sub-listas geradas à esquerda de $++$ são vazias, e as listas à direita tem tamanho $n-1$, $n-2$, $\cdots$, 1, e somente uma delas existe em um determinado tempo. A complexidade espacial *total*, considerando que a lista de entrada é impressa à medida que o resultado é computado é $\mathbb{O}(n)$, máxima. Na verdade, o espaco extra usado é constante, assim $\mathbb{R}(quick\ l) = \mathbb{k}$.

O pior caso para a complexidade espacial é quando o inverso da lista ordenada é a entrada de *quick*. Agora, as sublistas à esquerda de $++$ têm tamanho $n-1$, $n-2$, $\cdots$, 1, somente uma existindo em um determinado momento. As sub-listas à direita de $++$ são todas vazias, mas, *como o mecanismo de execução é preguiçoso*, elas não são computadas até que a necessidade obriga a tal. Isto significa que fechamentos contendo sub-listas de tamanho $n-1$, $n-2$, $\cdots$, 1, são retidos até que a computação de $++$ exija sua redução, o que implica numa complexidade espacial $\mathbb{O}(n^2)$.

Em média, se a árvore de $++$'s é balanceada, a complexidade espacial é dada pela fórmula $\phi(n) = \phi(n/2) + n$, ou $\mathbb{R}(quick\ l) = \mathbb{O}(2 * (\#l))$, o que é muito pobre comparado com o melhor algoritmo imperativo.

Observando que a causa deste comportamento é *preguiça*, nós podemos derivar uma solução de complexidade espacial $\mathbb{O}(n)$, forçando a redução das listas em todos os níveis, independentes delas estarem à esquerda ou à direita de $++$. Isto é feito em

$$\begin{aligned} quicker\ [\,] &= [\,] \\ quicker\ [a] &= [a] \\ quicker\ (a : x) &= split\ a [\,]\ [\,]\ x \\ split\ a\ left\ right\ [\,] &= quicker\ left\ ++\ a : quicker\ right \\ split\ a\ left\ right\ (b : x) &= split\ a\ (b : left)\ right\ x, \quad b\ <=\ a \\ &= split\ a\ left\ (b : right)\ x, \end{aligned}$$

que é $\mathbb{O}(n)$ em espaço no pior caso. Não faz sentido, em uma linguagem recursiva, eliminar recursão, a não ser em termos dos detalhes de implementação da linguagem. Assim sendo, não tentaremos melhorar a eficiência deste algoritmo ainda mais, apesar de o nosso arsenal de transformações ainda não ter se esgotado (ver [Meira 85t]). Em termos práticos, a definição acima é comparável à melhor definição imperativa, e este era o objetivo inicial deste exercício: mostrar que é possível, usando linguagens como KRC, ter todas as vantagens que podem ser tiradas da sua elegância semântica, sem necessariamente ter que usar algoritmos e definições impraticáveis.

8. Conclusão

Este artigo mostra uma das pontas do iceberg de programação funcional, a transformação de especificações em definições eficientes. Em alguns casos, esta tranformação não é necessária. Um exemplo disso é a definição de *perms* na Seção 3, que é teoricamente quase ótima.

Nos casos em que transformações são necessárias, elas são realizadas usando o mesmo formalismo (a linguagem de programação), mantendo o mesmo ambiente usado na especificação, o que certamente ajuda no desenvolvimento de programas. Apesar de termos usado um método informal neste artigo, todas as transformações usadas podem ser realizadas formalmente, passo a passo, acompanhadas de uma prova de correção de cada nova equação gerada. Dado à complexidade de tal exercício, ele é viável somente quando um sistema automático possa verificar a validade das transformações utilizadas. Tal sistema – que já existe para prova de correção de programas, LCF [Milner et al. 77] é um exemplo, poderia mesmo aplicar transformações elementares sem interferência do programador.

Finalizando, temos a dizer que, tendo sido ganho o argumento filosófico e psicológico em favor de linguagens funcionais como KRC e MIRANDA para desenvolvimento de software, em detrimento de FORTRAN e coisas do mesmo tipo, resta agora vencer a barreira que ainda impede o uso em larga escala de linguagens funcionais: eficiência e velocidade. Este trabalho é uma contribuição à solução do primeiro problema.

9. Agradecimentos

Este artigo foi composto em LaTeX, e impresso numa Canon LBP-10 Laser Printer, no Computing Laboratory da University of Kent at Canterbury. Gostaria de registrar meus agradecimentos ao Computing Lab por ter me dado acesso aos recursos acima.

Agradeço a Pedro Silveira pela ajuda na tradução de alguns termos técnicos para o Português.

Este trabalho teve o auxílio financeiro do CNPq, Proc. 200.510/81, e do Departamento de Eletrônica e Sistemas da UFPE.

Referências

Bjørner et al 80
BJØRNER, B. B., LOVEGREEN, H. H., BUNDGAARD, J., DOMMERGAARD, O. H., OEST, O. H., PEDERSEN, J. S. and SCHULTZ, L., *Towards a Formal Description of ADA*. ed.: D. B. Bjørner and O. N. Oest. Springer-Verlag, Berlin, (1980).

Boehm 81
BOEHM, B. W., *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, (1981).

Hoare 62
HOARE, C. A. R., Quicksort. *Comp. J.* **5**, (1962), pp. 10-15.

Honeywell 80
CII HONEYWELL BULL, *Reference Manual for the ADA Programming Language. Proposed Standard Document.* (1980).

Huet & Oppen 80
HUET, G. and OPPEN, D. C., Equations and Rewrite Rules: A Survey. *in Formal Language Theory, Perspectives and Open Problems*, ed.: Book, R. V. (1980), Acad. Press, New York,

Burstall 69
BURSTALL, R. M., Proving Properties of Programs by Structural Induction. *Comp. J.* **12**, (1), (Feb. 1969), pp. 41-48.

Meira 82
MEIRA, S. L., Sorting Algorithms in KRC: Implementation, Proof and Performance. *Comp. Lab. Rep. 14*, Computing Laboratory, University of Kent, Canterbury, UK, (1982).

Meira 84

MEIRA, S. L., Optimized Combinatoric Code for Applicative Language Implementation. *in Proc. of the VI Intl. Symp. on Programming*, Springer Verlag, *Lec. Notes in Comp. Sci.*, Heidelberg, (1984).

Meira 85t

MEIRA, S. L. *On the Efficiency of Applicative Algorithms. Ph. D. Thesis*, Computing Laboratory, University of Kent, Canterbury, UK, (1985).

Meira 85e

MEIRA, S. L. Efficient Algorithms in MIRANDA. *in preparation*, Dep. Elet. e Sistemas, UFPE, Recife-PE, Brasil, (1985).

Mendelson 79

MENDELSON, E., *Introduction to Mathematical Logic.* Van Nostrand, New York, (1979).

Milner et al. 77

MILNER, R., WADSWORTH, C. and GORDON, M., Edinburgh LCF. *CSR-11-77, Parts I and II*, Dep. Comp. Sci, University of Edinburgh, Edinburgh, Scotland, (1977).

Mycroft 81

MYCROFT, A., *Abstract Interpretation and Optimising Transformations for Applicative Programs. Ph. D. Thesis*, Dep. of Comp. Sci., Univ. of Edinburgh, Edinburgh, (1981).

Sedgewick 83

SEDGEWICK, R., *Algorithms.* Addison-Wesley, Reading, Mass., (1983).

Sleep & Holmstrom 82

SLEEP, M. R. and HOLMSTROM, S., *A Short Note Concerning Lazy Evaluation Rules of APPEND. Univ. of East Anglia Internal Report*, Norwich, U. K., (May 1982).

Strachey 72

STRACHEY, C., Varieties of Programming Language. *Infotech State of the Art Report on "High Level Languages" 7*, (1972).

Turner 79a

TURNER, D. A., A New Implementation Technique for Applicative Languages. *Soft. Pract. and Exp.* **9**, (1979), pp. 31-49.

Turner 81a

TURNER, D. A., *Aspects of the Implementation of Programming Languages. Ph. D. Thesis*, Brasenose College, Oxford University, Oxford, (1981).

Turner 82b

TURNER, D. A., Recursion Equations as a Programming Language. *in Functional Programming and its Applications, An Advanced Course*, C. U. P., Cambridge, UK, (1982).

Turner 84

TURNER, D. A., Functional Programs as Executable Specifications. *Phil. Trans. Royal. Soc. of London* **312**, (1522), (1984), pp. 363-388.

Wadsworth 71

WADSWORTH, C., *Semantics and Pragmatics for the Lambda Calculus. Ph. D. Thesis*, Dept. of Mathematics, Oxford Univ., Oxford, (1971).

Wirth 76

WIRTH, N. Algorithms + Data Structures = Programs. Prentice-Hall, Eng. Cliffs, NJ, (1976).

ESTUDO EMPÍRICO DE UMA NOVA IMPLEMENTAÇÃO PARA O ALGORITMO DE CONSTRUÇÃO DE ÁRVORE PATRICIA

L. C. A. ALBUQUERQUE[1], N. ZIVIANI[2]

SUMÁRIO

PATRICIA é um algoritmo eficiente para casamento de cadeias. Este trabalho tem por objetivo apresentar um algoritmo melhorado para construção de árvores Patricia. O algoritmo proposto é comparado empiricamente com os algoritmos mais conhecidos de construção de árvores Patricia, obtendo resultados bem favoráveis ao novo algoritmo.

ABSTRACT

PATRICIA is an efficient algorithm for string matching. The objective of this paper is to present a better algorithm for Patricia tree construction. The proposed algorithm is compared empirically to the well known algorithms for Patricia tree construction, and favorable results are obtained.

[1] Bel. em Matemática (UFV, 1974), Aluno de Pós-Graduação em Ciência da Computação (UFMG); estruturas de dados, linguagens e sistemas de programação; Departamento de Matemática; UFV, 36570 Viçosa, MG. Tel.: (031) 891-1790, r. 368.

[2] Engº Mecânico (UFMG, 1971), Mestre em Informática (PUC/RJ, 1976), Ph.D. em Ciência da Computação (Univ. Waterloo, 1982); engenharia de software, estruturas de dados, análise de algoritmos; Departamento de Ciência da Computação, ICEx, UFMG, Caixa Postal 702, 30000 Belo Horizonte, MG. Tel.: (031) 443-4088.

1. INTRODUÇÃO

PATRICIA é a abreviatura de *Pratical Algorithm To Retrieve Information Coded In Alphanumeric* (Algoritmo Prático para Recuperar Informação Codificada em Alfanumérico). Este algoritmo foi originalmente criado por MORRISON [1968] num trabalho de casamento de cadeias (*string matching*), aplicado em recuperação de informação em arquivos de grande porte. KNUTH [1973] deu um novo tratamento ao algoritmo, reapresentando-o de forma mais clara como um caso particular de pesquisa digital, essencialmente, um caso de árvore *trie* binária.

Além disso KNUTH [1973] apresentou explicitamente um algoritmo de pesquisa e propôs um algoritmo de inserção (construção de árvore Patricia) como exercício, dando sugestão de como elaborá-lo na parte de respostas aos exercícios. Ele apresentou também um estudo analítico do algoritmo de pesquisa, obtendo o número de inspeções de bits para uma pesquisa aleatória com sucesso e para uma pesquisa sem sucesso; esses valores são, respectivamente, $\log_2 N + 0.33275$ e $\log_2 N - 0.31875$, onde N é o número de chaves armazenadas na árvore.

SEDGEWICK [1983] apresentou novos algoritmos de pesquisa e de inserção baseados nos algoritmos propostos por Knuth. GONNET [1984a] propôs também outros algoritmos de pesquisa e de inserção usando recursividade e uma estrutura de dados ligeiramente diferente para representar a árvore.

O objetivo deste trabalho é o de apresentar uma melhoria no algoritmo de inserção cuja característica principal é a de conseguir construir a árvore Patricia utilizando o menor número de inspeções de bits dentre os algoritmos mais conhecidos. Para mostrar esta característica do algoritmo, realizou-se um estudo experimental, onde foram comparados quatro algoritmos de inserção:

(1) Algoritmo de Sedgewick (SEDGEWICK [1983], p. 222),
(2) Algoritmo de Gonnet (GONNET [1984a], pp. 110-111),
(3) Algoritmo Melhorado de Gonnet (GONNET [1984b]),
(4) Algoritmo Proposto.

As medidas de complexidade consideradas foram:

I(N) - o número total de inspeções de bits necessário para construir uma árvore com N chaves;
C(N) - o número de comparações entre bits de chaves necessário para construir uma árvore com N chaves;
W(N) - o número de inspeções de bits necessário para traçar o caminho de pesquisa e inserção durante a construção de uma árvore com N chaves;
E(N) - o número de nodos de uma árvore com N chaves.

Dentre os algoritmos estudados, verificou-se que os algoritmos (1) e (4) acima são os que apresentam melhor desempenho quanto ao número total I(N) de inspeções de bits, com o al-

goritmo (4) apresentando ainda uma redução de mais de 20% sobre o algoritmo (1) no número I(N). Cabe observar que a inspeção de bits é a operação mais cara dentre as operações dos algoritmos considerados.

2. PESQUISA DIGITAL

A pesquisa digital é baseada na representação das chaves como uma seqüência de caracteres ou de dígitos. A grosso modo, o método de pesquisa digital é realizado da mesma forma que uma pesquisa em dicionários que possuem aqueles "índices de dedo". Com a primeira letra da palavra são determinadas todas as páginas que contêm as palavras iniciadas por aquela letra.

Os métodos de pesquisa digital são particularmente vantajosos quando as chaves são grandes e de tamanho variável. No problema de casamento de cadeias trabalha-se com chaves semi-infinitas, isto é, sem limitação explícita quanto ao tamanho delas. Um aspecto interessante quanto aos métodos de pesquisa digital é a possibilidade de localizar todas as ocorrências de uma determinada cadeia em um texto, com tempo de resposta logarítmico em relação ao tamanho do texto.

Este aspecto é particularmente importante em aplicações que utilizam grandes bancos de dados, tais como os utilizados em sistemas legislativos e judiciários, jornalismo, informações técnico-científicas, informações policiais, informações mercadológicas, bibliotecas e centros de informação em geral. Nesses tipos de arquivos apenas adições são realizadas, e as consultas são imprevisíveis. Como exemplo, alguém poderia necessitar de uma relação dos artigos de periódicos científicos que tratam do assunto "bancos de dados não estruturados", ou então a relação dos acidentes de tráfego causados por embriaguez em regiões urbanas durante os fins de semana. A apresentação de algoritmos eficientes para recuperação de informação utilizando pesquisa digital em bancos de dados não estruturados é tratada em detalhes em ALBUQUERQUE, GONNET e ZIVIANI [1985].

A seguir apresentam-se os métodos de pesquisa digital mais utilizados.

2.1. *Trie*

Uma *trie* é uma árvore M-ária cujos nodos são vetores de M componentes com campos correspondentes aos dígitos ou caracteres que formam as chaves. Cada nodo no nível i representa o conjunto de todas as chaves que começam com a mesma seqüência de i dígitos ou caracteres. Este nodo especifica uma ramificação com M caminhos dependendo do (i+1)-ésimo dígito ou carácter de uma chave.

Supondo as chaves como seqüência de bits, isto é, M = 2, o algoritmo de pesquisa digital em *trie* é semelhante ao de pesquisa em árvore binária, exceto que, ao invés de se caminhar na árvore de acordo com o resultado de comparação entre chaves, caminha-

se de acordo com os bits da chave. Em 1960, Fredkin chamou este método de *trie*, que corresponde a um pedaço da palavra *retrieval* (KNUTH [1973], p. 481).

A Figura 2.1.1. mostra uma *trie* construída a partir das seguintes chaves de 6 bits:

B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000

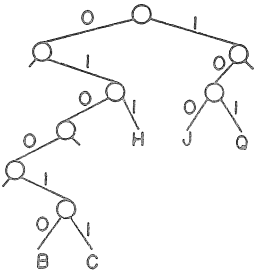


Fig. 2.1.1

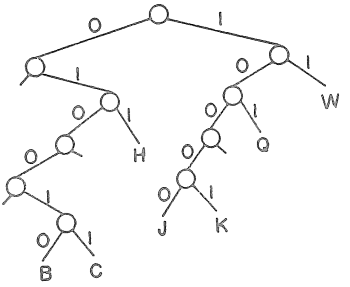


Fig. 2.1.2

Para construir uma *trie* faz-se uma pesquisa na árvore com a chave a ser inserida. Se o nodo externo em que a pesquisa terminar for vazio, cria-se um novo nodo externo nesse ponto contendo a nova chave, como ilustra a inserção da chave W = 110110 na Figura 2.1.2. Se o nodo externo contiver uma chave, cria-se um ou mais nodos internos cujos descendentes conterão a chave já existente e a nova chave. A Figura 2.1.2 ilustra a inserção da chave K = 100010 que envolve repor J por um novo nodo interno cuja sub-árvore esquerda é outro novo nodo interno cujos filhos são J e K, porque estas chaves possuem os mesmos bits até a quinta posição.

O formato das *tries*, diferentemente das árvores binárias comuns, não depende da ordem em que as chaves são inseridas e sim da estrutura das chaves através da distribuição de seus bits. Uma grande desvantagem das *tries* é a formação de caminhos de uma só direção para chaves com um grande número de bits em comum. Por exemplo, se duas chaves diferirem somente no último bit, elas formarão um caminho cujo comprimento é igual ao tamanho delas, não importando quantas chaves existem na árvore. Veja o caminho gerado pelas chaves B e C na Figura 2.1.1.

2.2. Patricia

O algoritmo para construção de árvore Patricia (MORRISON [1968]) é baseado no método de pesquisa digital, mas sem apresentar o inconveniente citado para o caso das *tries*. O problema de caminhos de uma só direção é eliminado por meio de uma solução simples e elegante: cada nodo interno da árvore contém o índice do bit a ser testado para decidir qual rumo tomar. A Figura 2.2.1 apresenta a árvore Patricia gerada a partir das chaves dadas acima.

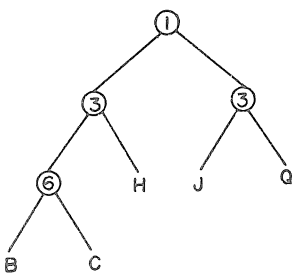


Fig. 2.2.1

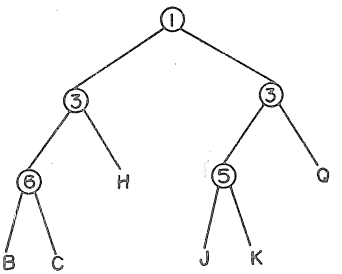


Fig. 2.2.2

Para inserir a chave K = 100010 na árvore da Figura 2.2.1, a pesquisa inicia pela raiz e termina quando se chega ao nodo externo contendo J. Os índices dos bits nas chaves estão ordenados da esquerda para a direita. Assim, o bit de índice 1 de K é 1, indicando a sub-árvore direita, e o bit de índice 3 indica a sub-árvore esquerda que, neste caso, é um nodo externo. Isto significa que as chaves J e K mantêm o padrão de bits 1x0xxx, assim como qualquer outra chave que seguir este caminho de pesquisa. Um novo nodo interno repõe o nodo J, e este juntamente com o nodo K serão os nodos externos descendentes. O índice do novo nodo interno é dado pelo primeiro bit diferente das duas chaves em questão, que é o bit de índice 5. Para determinar qual será o descendente esquerdo e o direito, é só verificar o valor do bit 5 de ambas as chaves, conforme mostrado na Figura 2.2.2.

A inserção da chave W = 110110 ilustra um outro aspecto. A pesquisa sem sucesso na árvore da Figura 2.2.2 é realizada de maneira análoga. Os bits das chaves K e W são comparados a partir do primeiro para determinar em qual índice eles diferem, sendo, neste caso, os de índice 2. Portanto o ponto de inserção agora será no caminho de pesquisa entre os nodos internos de índice 1 e 3. Cria-se aí um novo nodo interno de índice 2, cujo descendente direito é um nodo externo contendo W e cujo descendente

esquerdo é a sub-árvore de raiz de índice 3, conforme ilustra a Figura 2.2.3.

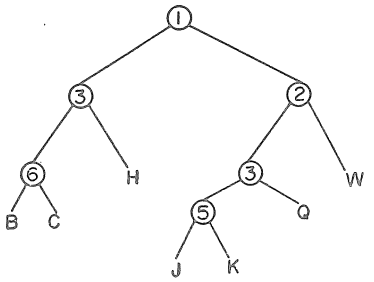


Fig. 2.2.3

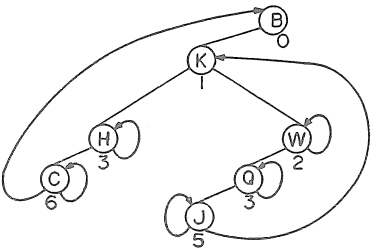


Fig. 2.2.4

O método delineado acima, assim como o método da *trie*, usa dois tipos distintos de nodos. Os nodos internos contêm o índice do bit a ser inspecionado e os nodos externos contêm somente as chaves. Ao invés de armazenar as chaves nos nodos externos, existe uma outra opção que é a de manter somente um apontador para outra estrutura que contenha as chaves fisicamente. Isto é interessante principalmente no caso de as chaves serem longas cadeias de caracteres.

Em algumas linguagens de programação pode ser complicado definir uma estrutura de dados com dois tipos de nodos diferentes. Para contornar este problema KNUTH [1973] sugere a criação de mais um campo nos nodos internos para conter as chaves. Os nodos externos são embutidos nos nodos internos, fazendo com que todos tenham a mesma estrutura. Neste caso as chaves nos nodos não serão usadas ao longo do caminho de pesquisa. Ao chegar a um ponto de um nodo externo, a chave é localizada através de um apontador dirigido para cima, conforme ilustra a Figura 2.2.4. Embora o tamanho de cada nodo aumente, isto reduz à metade o número de nodos da árvore, pois os nodos externos não existem mais explicitamente. Observe na Figura 2.2.4 que as chaves estão dentro dos nodos e os índices estão abaixo. Além disso há um nodo-cabeça cujo índice foi considerado como 0 (zero).

Agora aparece um pequeno problema: como diferenciar um apontador puro para um nodo interno de um apontador para cima simulando a ocorrência de um nodo externo? KNUTH [1973] utiliza mais dois campos extras de 1 bit cada para indicar se o respectivo apontador para a a raiz da sub-árvore está apontando para um nodo interno ou um nodo externo (para cima). E esta é a única solução razoável que poderia ser tomada por Knuth, pois o campo de índice formulado por ele não contém o real valor do índice, e sim o incremento que deve ser dado ao valor corrente do índice para obter o novo índice.

Isto tem a vantagem de serem necessários menos bits no campo de índice, compensando, assim, os dois bits adicionais para discriminar os apontadores das sub-árvores.

SEDGEWICK [1983] adotou outra solução a partir da observação de que os índices de uma mesma sub-árvore aparecem ordenadamente em relação ao nível do nodo. Neste caso o índice de um nodo de nível i é menor do que o de um nodo de nível j, desde que i < j. Logo, testando os índices dos nodos do caminho de pesquisa, é possível saber se se chegou a um "nodo externo" ou não.

3. ALGORITMOS PATRICIA

A seguir são apresentados dois algoritmos conhecidos para construção de árvore Patricia, uma proposta de melhoria de um deles, e uma nova proposta de algoritmo que é mais eficiente que os três primeiros.

3.1. Algoritmo de Sedgewick

O Algoritmo de Sedgewick (SEDGEWICK [1983], p. 222) usa a estrutura de dados para representar a árvore com os nodos externos implícitos. Neste caso a medida de espaço é E(N) = N, onde N é o número de chaves da árvore.

A árvore é inicializada com um nodo-cabeça contendo a primeira chave a ser inserida, o campo de índice igual a 0 (zero) e o apontador de sub-árvore esquerda dirigido para o próprio nodo. Considerando uma árvore com n chaves, a inserção de uma nova chave k segue os passos abaixo:

1. Desce-se pela árvore da maneira já descrita na Seção 2.2 até chegar a uma posição de um "nodo externo", o qual não existe explicitamente na árvore, e sim embutido em algum nodo acima.
2. Os bits da chave k' do nodo determinado no passo 1 são comparados com os respectivos bits da chave k, a partir do primeiro, até chegar a um índice cujos bits difiram. Se todos os bits forem iguais, a chave k já se encontra na árvore e o algoritmo termina.
3. Desce-se novamente pela árvore a partir da raiz até chegar a um nodo cujo campo de índice seja maior que o índice i encontrado no passo 2, ou até chegar a uma posição de "nodo externo". Este é o ponto de inserção do novo nodo contendo a chave k e o campo de índice igual a i. A inserção propriamente é feita como descrito na Seção 2.2.

É bom notar que, em todos estes três passos, são realizados inspeções de bits. E, além disso, fazem-se duas descidas pela árvore. Para efeito de estruturação do Algoritmo de Sedgewick, considerou-se que cada chave possui um bit 0 virtual na posição 0, ou seja, à esquerda do bit de índice 1.

3.2. Algoritmo de Gonnet

GONNET [1984a, pp. 110-111] formula seu algoritmo de forma recursiva e utiliza os nodos externos explicitamente. Neste caso a medida de espaço é E(N) = 2N - 1, onde N é o número de chaves armazenadas na árvore.

A árvore é inicializada vazia. Em seguida, qualquer chave k é inserida, partindo da raiz, segundo o esquema:

1. Se a sub-árvore em questão é vazia, é criado um novo nodo externo contendo a chave k (isto ocorre somente na inserção da primeira chave) e o algoritmo termina. Caso contrário, é obtida uma chave k' de qualquer nodo externo da sub-árvore (por exemplo, o nodo externo mais à esquerda).
2. Os bits da chave k' obtida no passo 1 são comparados, a partir do primeiro, com os bits correspondentes da chave k até chegar a um índice i cujos bits difiram. Se todos forem iguais, a chave k já se encontra na árvore e o algoritmo termina.
3. Se a sub-árvore em questão for um nodo externo ou o índice da sua raiz for maior que o índice i encontrado no passo 2, inserem-se, neste ponto, um novo nodo interno com índice igual a i e um novo nodo externo contendo a chave k, da forma descrita na Seção 2.2.
4. Caso contrário, executam-se recursivamente os passos 1, 2 e 3 novamente, considerando a sub-árvore esquerda ou a direita, de acordo com o bit da chave k de índice indicado pelo nodo da sub-árvore corrente.

Neste caso desce-se somente uma vez pela árvore. Mas, em toda instância, tem-se que acessar um nodo externo qualquer da sub-árvore corrente (passo 1), havendo inspeções de bits (passo 2), além da inspeção usual que é realizada para descer pela árvore.

3.3. Algoritmo Melhorado de Gonnet

No passo 2 do Algoritmo de Gonnet, observou-se que não seria necessário inciar a comparação dos bits a partir do primeiro, a cada ativação da chamada recursiva. Isso porque os bits já foram comparados até o índice i da chamada recursiva anterior (GONNET [1984b]). Assim sendo, basta iniciar a comparação a partir desse índice na ativação corrente. É digno de nota que, para cada comparação de bits, são necessárias duas inspeções de bits.

3.4. Algoritmo Proposto

O algoritmo proposto também é recursivo e utiliza os nodos externos explicitamente. Entretanto, com pequenas alterações é possível implementá-lo com os nodos externos embutidos. A árvore é inicializada com um nodo-cabeça interno contendo 0 (zero) no campo de índice e o apontador de sub-árvore esquerda nulo. Neste caso a medida de espaço é E(N) = 2N, sendo N o número de chaves da árvore.

Cada chave k é inserida de acordo com os passos abaixo, partindo da raiz:

1. Se a sub-árvore corrente é vazia, então é criado um nodo externo contendo a chave k (isto ocorre somente na inserção da primeira chave) e o algoritmo termina.
2. Se a sub-árvore corrente é simplesmente um nodo externo, os bits da chave k são comparados, a partir do primeiro, com os bits correspondentes da chave k' deste nodo externo até encontrar um índice i cujos bits difiram. (Se todos forem iguais, a chave já se encontra na árvore e o algoritmo termina). Depois são criados um nodo interno e um nodo externo: o primeiro contendo o índice i e o segundo, a chave k. O nodo interno é ligado ao externo pelo apontador de sub-árvore esquerda ou direita, dependendo se o bit de índice i da chave k seja 0 ou 1 respectivamente.
3. Caso contrário, ou seja, se a raiz da sub-árvore corrente for um nodo interno, vai-se para a sub-árvore indicada pelo bit da chave k de índice do nodo corrente, de forma recursiva.
4. O caminho de inserção é percorrido novamente de baixo para cima, subindo com o par de nodos criados no passo 2 até chegar a um nodo interno cujo índice seja menor que o índice determinado também no passo 2. Este é o ponto de inserção e o par de nodos é inserido de acordo com o que foi visto na Seção 2.2.

Este algoritmo utiliza o menor número de operações de inspeção de bits em relação aos demais, por causa do esquema utilizado no passo 4, que evita percorrer a árvore uma segunda vez, aproveitando o mecanismo da recursividade. Da mesma forma que o Algoritmo de Sedgewick, considerou-se que cada chave tem um bit 0 virtual à esquerda do bit de índice 1, para efeito de estruturação do algoritmo. Para maiores detalhes, veja o Algoritmo Proposto descrito em Pascal no Apêndice.

4. RESULTADOS EXPERIMENTAIS

O desempenho dos algoritmos de construção de árvore Patrícia foi determinado por meio do seguinte experimento: foram geradas N chaves distintas e aleatórias, cada uma com 32 bits uniformemente distribuídos. Utilizou-se um gerador de números aleatórios baseado no método de congruência multiplicativo. As chaves foram inseridas uma a uma, em uma árvore inicialmente vazia. Durante as inserções foram feitos os cálculos das medidas de complexidade I(N), C(N) e W(N) definidas na Seção 1. O mesmo conjunto de N chaves foi submetido a todos os quatro algoritmos. O experimento foi repetido um número suficiente de vezes para tamanhos diferentes de N.

As Tabelas 4.1, 4.2, 4.3 e 4.4 apresentam os resultados médios obtidos para árvores contendo diversos números de chaves. Os intervalos de confiança para as médias foram ao nível de 95% de probabilidade.

Tabela 4.1. Algoritmo de Sedgewick

N	I(N)	C(N)	W(N)
50	1025 ± 4	277 ± 2	472 ± 2
100	2451 ± 6	654 ± 3	1143 ± 3
500	16895 ± 13	4430 ± 5	8034 ± 7
1000	37792 ± 22	9863 ± 8	18066 ± 14
5000	235282 ± 78	60883 ± 34	113516 ± 38
10000	510495 ± 119	131724 ± 45	247048 ± 58

Tabela 4.2. Algoritmo de Gonnet

N	I(N)	C(N)	W(N)
50	2049 ± 17	935 ± 9	178 ± 1
100	5507 ± 29	2526 ± 14	455 ± 2
500	47913 ± 89	22241 ± 43	3431 ± 5
1000	116488 ± 180	54314 ± 87	7861 ± 8
5000	861190 ± 689	405153 ± 338	50884 ± 23
10000	1995768 ± 888	941991 ± 432	111786 ± 35

Tabela 4.3. Algoritmo Melhorado de Gonnet

N	I(N)	C(N)	W(N)
50	1088 ± 5	455 ± 2	178 ± 1
100	2673 ± 8	1109 ± 3	455 ± 2
500	19155 ± 17	7862 ± 7	3431 ± 5
1000	43307 ± 30	17723 ± 12	7861 ± 8
5000	274418 ± 90	111767 ± 39	50884 ± 23
10000	598806 ± 149	243510 ± 62	111786 ± 35

Tabela 4.4. Algoritmo Proposto

N	I(N)	C(N)	W(N)
50	799 ± 4	277 ± 2	246 ± 1
100	1898 ± 6	654 ± 3	590 ± 1
500	12965 ± 11	4430 ± 5	4105 ± 3
1000	28934 ± 18	9863 ± 8	9208 ± 6
5000	179400 ± 74	60883 ± 34	57634 ± 17
10000	388711 ±101	131724 ± 45	125263 ± 26

O Algoritmo de Gonnet realiza o menor número W(N) de inspeções de bits para dirigir o caminhamento pela árvore. Isto porque ele faz somente uma descida pela árvore, parando o caminhamento já no ponto de inserção. Porém, a cada nível do caminho, são realizadas comparações de bits como mostram os altos valores de C(N). O Algoritmo Melhorado de Gonnet diminui os valores de C(N), utilizando o mecanismo descrito na Seção 3.3.

O Algoritmo de Sedgewick e o Proposto realizam o mesmo número de C(N) de comparações de bits. Ambos fazem esta operação somente quando chegam a um nodo externo, evitando ter que fazê-lo a cada nível do caminho de descida. O Algoritmo Proposto reduz ainda mais o número total I(N) de inspeções de bits em relação ao Algoritmo de Sedgewick, pois este desce uma segunda vez pela árvore e o Algoritmo Proposto aproveita o caminho já traçado pela descida inicial. Isto é refletido pela redução dos valores de W(N).

Para todos os algoritmos estudados vale a relação:

$$I(N) = 2C(N) + W(N).$$

A operação de inspeção de bits cuja medida é dada por I(N) constitui a operação mais cara dos algoritmos Patricia. Nas aplicações práticas desses algoritmos, a rotina responsável pelas inspeções de bits é que deverá fazer os acessos às chaves, as quais certamente estarão armazenadas em memória secundária. Portanto, o fator predominante de custo será a operação de inspeção de bits. Dos quatro algoritmos estudados o que apresenta o menor I(N) é o Algoritmo Proposto. Daí ser ele o mais eficiente em tempo.

5. CONCLUSÕES

As árvores Patricia são adequadas quando se trabalha com chaves de tamanho muito grande e variável. Chaves desse tipo aparecem em problemas de casamento de cadeias, os quais têm aplicação prática em grandes bancos de dados não estruturados sob a forma de informações textuais.

O número esperado de inspeções de bits para pesquisa em árvores Patricia foi determinado

analiticamente por KNUTH [1973], obtendo valores $O(\log_2 N)$. Por outro lado, o custo de construção de uma árvore com N chaves nunca foi estudado analiticamente. Este trabalho propõe mais um algoritmo de inserção. O algoritmo proposto foi estudado empiricamente junto com outros dois algoritmos conhecidos. Os resultados experimentais mostraram que o algoritmo proposto é o que faz menos inspeções de bits, sendo esta a operação crucial durante a construção de uma árvore Patricia.

6. BIBLIOGRAFIA

ALBUQUERQUE, L. C. A.; GONNET, G. H. e ZIVIANI, N. [1985]. "Recuperação Eficiente de Informação em Bancos de Dados Não Estruturados". Série de Monografias do Departamento de Ciência da Computação, UFMG, Belo Horizonte (a ser publicado).

GONNET, G. H. [1984a]. *Handbook of Algorithms and Data Structures*. Addison-Wesley, Reading, Massachusetts.

GONNET, G. H. [1984b]. Comunicação pessoal.

KNUTH, D. E. [1973]. *The Art of Computer Programming; vol. 3: Sorting and Searching*. Addison-Wesley, Reading, Massachusetts.

MORRISON, D.R. [1968]. "PATRICIA - Pratical Algorithm to Retrieve Information Coded in Alphanumeric". *Journal of the ACM*, 15:4 (October), 514-534.

SEDGEWICK, R. [1983]. *Algorithms*. Addison-Wesley, Reading, Massachusetts.

APÊNDICE

ALGORITMO PROPOSTO

1. Definição da Estrutura de Dados

```
const D = "nº de bits de cada chave";  {depende de ChaveTipo}
type ChaveTipo = "a definir; depende da aplicação";
     IndexAmp = 0..D;
     NodoTipo = (interno, externo);
     Arvore = ↑PatNodo;
     PatNodo = record
                  case nt : NodoTipo of
                  interno :
                     (index : IndexAmp;
                      esq, dir : Arvore);
                  externo :
                     (chave : ChaveTipo)
               end;
     Dib = 0..1;
```

2. Funções e Procedimentos Auxiliares

```
function Bit (i : IndexAmp; k : ChaveTipo) : Dib;
{Retorna o i-ésimo bit, a partir da esquerda, da chave k.
 A implementação abaixo é dada a título de exemplo. Uma
 implementação mais eficiente para uso em programas produto
 deve ser feita em linguagem simbólica (assembler), dependendo
 assim da máquina hospedeira.}
var c, j : integer;
begin
   if i = 0 then Bit := 0 else
   begin
      c := ord(k);
      for j := 1 to D-i do c := c div 2;
      Bit := c mod 2
   end
end {Bit};

function EData (p : Arvore) : boolean;
{Verifica se p↑ é nodo externo.}
begin
   EData := p↑.nt = externo
end {EData};

procedure CrieNodos (k, ka : ChaveTipo; var v : Arvore;
                     var h : boolean);
{Cria um par de nodos: um interno e outro externo. O nodo
 interno conterá o valor do índice do bit em que as chaves
 k e ka diferem. O nodo externo conterá a chave k a ser
 inserida. O nodo interno é ligado ao nodo externo de acordo
 com o bit em que as chaves diferem. O endereço do nodo
 interno retorna em v. h indica se a inserção já foi feita.}
var i : integer; p, q : Arvore;
    b : boolean;
begin {CrieNodos}
   i := 1; b := true;
   while b and (i ≤ D) do
      if Bit(i,k) = Bit(i,ka) then i := i + 1
      else b := false;
   if i > D then
   begin {k já se encontra na árvore}
      h := true; erro
   end else
   begin
      h := false;
      new (p, interno); new (q, externo);
      with p↑ do
      begin nt := interno; index := i;
         if Bit(i,k) = 0 then esq := q
                         else dir := q
      end;
      with q↑ do
      begin nt := externo; chave := k
      end;
      v := p
   end
end {CrieNodos};
```

3. Algoritmo de Pesquisa

```
procedure Pesquise (k : ChaveTipo; t : Arvore);

begin
   if EData(t) then
      if k = t↑.chave then Encontrado(t)
                      else NaoEncontrado(k)
   else
      if Bit (t↑.index, k) = 0 then Pesquise (k, t↑.esq)
                               else Pesquise (k, t↑.dir)
end {Pesquise};
```

4. Algoritmo de Inserção

```
procedure Insira (k : ChaveTipo; var t, u : Arvore;
                  var h : boolean);

{k é a chave a ser inserida;
 t contém o endereço da raiz da sub-árvore corrente; na
   primeira chamada t = raiz;
 u contém o endereço do nodo interno ligado a um nodo externo
   contendo a chave k a ser inserida. Este par de nodos sobe
   pelo caminho de pesquisa até chegar ao ponto de inserção.
 h indica se já foi realizada a inserção ou não do par de
   nodos; se h = true, a inserção já se realizou.}

begin {Insira}
   if t = nil then {insere a primeira chave}
   begin
      h := true;
      new (t, externo);
      with t↑ do
      begin nt := externo; chave := k
      end
   end else
   if EData(t) then CrieNodos (k, t↑.chave, u, h) else
   if Bit (t↑.index, k) = 0 then
      begin
         Insira (k, t↑.esq, u, h);
         if not h then
            if t↑.index < u↑.index then
            begin {insere o par de nodos cujo endereço é u}
               h := true;
               if Bit(i,k) = 0 then u↑.dir := t↑.esq
                               else u↑.esq := t↑.esq;
               t↑.esq := u
            end
      end
   else
      begin
         Insira (k, t↑.dir, u, h);
         if not h then
            if t↑.index < u↑.index then
            begin {insere o par de nodos cujo endereço é u}
               h := true;
               if Bit(i,k) = 0 then u↑.dir := t↑.dir
                               else u↑.esq := t↑.dir;
               t↑.dir := u
            end
      end
end {Insira};
```

LES GRAPHES ADJOINTS DES GRAPHES SIMPLES AVEC $\Delta(G) \geq 4$ SONT ARÊTE-RECONSTRUCTIBLES

O. MEZA HOUTTEMAN#

RÉSUMÉ

Cet article a trait au problème de reconstruction de graphes. On démontre que la classe des graphes adjoints des graphes simples de degré maximum ≥ 4 est arête-reconstructible, c'est-à-dire, tout graphe dans cette classe est caracterisé par la liste de ses sous-graphes propres maximaux non-étiquetés.

ABSTRACT

This paper concern the graph reconstruction problem. We show that the Line graphs of graphs with maximum degree 4 are edge-reconstructibles, i.e., every graph in this class can be caracterised by its maximal proper non-labeled sub-graphs.

Licenciado en Computación (U.C.V.,1978), Doctor de tercer ciclo en Investigación de Operaciones (Universidad de Grenoble, Francia 1983). Profesor en la Universidad Simón Bolívar, Caracas, Venezuela. Area de interés: Optimización Combinatoria, Teoría de grafos, Complejidad Algorítmica.

LES GRAPHES ADJOINTS DES GRAPHES SIMPLES AVEC $\Delta(G) \geq 4$ SONT ARÊTE-RECONTRUCTIBLES:

Auteur: Oscar José Meza Houtteman
UNIVERSITE SIMON BOLIVAR
CARACAS. VENEZUELA.

Dans la suite tous les graphes considérés seront des graphes simples. Nous dénotons un graphe simple par $G=(V,E)$.

Definition 1: Un sous-graphe induit de G par $U \subseteq V$, c'est un graphe $G[U]$ dont les sommets sont ceux de U et dont les arêtes sont les arêtes de G ayant leurs deux extrémités dans U.

Definition 2: Un sous-graphe partiel ou sous-graphe de G est un graphe (U,F) avec $U \subseteq V$, $F \subseteq E$ et $\forall e=[x_1,x_2] \in F$: $x_1 \in U$ et $x_2 \in U$.

Definition 3: Le graphe adjoint d'un graphe G est le graphe L(G) dont les sommets sont les arêtes de G, deux sommets étant reliés dans L(G) si et seulement si les arêtes correspondantes dans G ont une extrémité commune.

Definition 4: Soit (G_e) la liste des sous-graphes propres maximaux non-étiquetés de G. On dit qu'un graphe H est une arête-reconstruction de G si la liste des H_e est égale à celle de G. On dit que G est arête-reconstructible si toute arête-reconstruction de G est isomorphe à G. Une classe de graphes est arête-reconnaissable si toute arête-reconstruction d'un graphe dans la classe appartient à la classe. Une classe de graphes est arête-recontructible si tout graphe dans la classe est arête-reconstructible.

Dans (1), Harary a formulé une conjecture analogue a celle d'Ulam: Tout graphe G avec $|E| \geq 4$ est arête-reconstructible.

Dans cet article on démontre que la classe $\mathcal{H}$ des graphes adjoints des graphes simples de degré maximum ≥ 4 ($\Delta(G) \geq 4$) vérifient la conjecture d'Harary.

Pour démontrer ce résultat nous utiliserons le lemme de Kelly pour l'arête-reconstruction (Bondy (2)):

Lemme 1: Soient F et G deux graphes tels que $|E(F)| < |E(G)|$. Le nombre $n(F,G)$ de sous-graphes partiels de G isomorphes à F est arête-reconstructible. Si G ne possède pas de sommets isolés et $|V(F)| < |V(G)|$ alors le nombre $s(F,G)$ de sous-graphes induits de G isomorphes à F est arête-reconstructible.

Démonstration:

1) Chaque sous-graphe partiel F de G est compté $|E(G)| - |E(F)|$ fois dans la somme: $\sum_{e \in E(G)} n(F,G_e)$. Ainsi:

$$n(F,G) = \sum_{e \in E(G)} n(F,G_e) \;/\; (|E(G)| - |E(F)|)$$

2) Bondy a montré que si G ne possède pas de sommets isolés alors la liste des G_v (les sous-graphes induits propres maximaux non-étiquetés de G) est arête-reconstructible. Donc, étant donnée cette liste on a:

$$s(F,G) = \sum_{v \in V(G)} s(F,G_v) \;/\; (\, V(G) - V(F) \,)$$

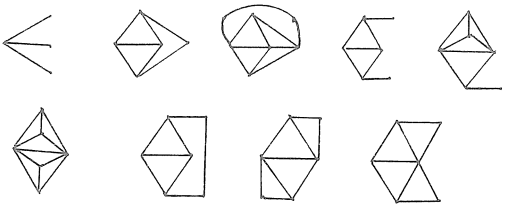
C.Q.F.D.

Greenwell (3) a arête-reconstruit le nombre de sommets isolés d'un graphe. Par conséquent, lorsqu'on reconstruit un graphe G', on lui enlève tout d'abord les sommets isolés. De cette facon, toute arête-reconstruction du graphe G, obtenu à partir de G' en lui enlevant les sommets isolés, ne comportera pas de sommets isolés. En plus, si G est arête-reconstructible alors G' est arête-reconstructible. D'après cette remarque, il suffit de démontrer que les $G \in \mathcal{H}$ sans sommets isolés sont arête-reconstructibles.

Lemme 2: (Caracterisation des graphes adjoints par des sous-graphes exclus)

G est un graphe adjoint si et seulement si aucun graphe de la liste (1) n'est un sous-graphe induit de G:

liste (1)



Soit $\mathcal{H} = \mathcal{G}' \cup \mathcal{G}$. Où:

- $\mathcal{G}'$ est la classe des graphes adjoints des graphes avec $\Delta(G) \geq 5$.
- $\mathcal{G}$ est la classe des graphes adjoints des graphes avec $\Delta(G) = 4$.

Si $G \in \mathcal{G}'$ alors G contient une 5-clique.

Lemme 3: $\mathcal{G}'$ est arête-reconnaissable.

Démonstration:

Soit $G \in \mathcal{G}'$. Il faut montrer que toute arête-reconstruction H de G est un graphe adjoint comportant une 5-clique:

1) Si G est isomorphe à K_5 alors K_5 est la seule arête-reconstruction de G car K_5 est arête-reconstructible (Harary (4)).

2) Si G n'est pas isomorphe à K_5:

- G comporte un sous-graphe sommet propre isomorphe à K_5. Donc, d'après le lemme 1, H contient une 5-clique.

- Pour montrer que H est un graphe adjoint, il suffit de montrer qu'il ne comporte pas de sous-graphe induit isomorphe à un des graphes de la liste (1) donnée plus haut: H ne peut pas etre isomorphe à un des graphes de la liste (1) car H comporte une 5-clique; par conséquent, si H contenait un des graphes de la liste (1), alors ce graphe serait un sous-graphe propre de H. Donc, d'après le lemme 1, il serait aussi sous-graphe de G ce qui est impossible car G .

C.Q.F.D.

Proposition 1:

$\mathcal{G}'$ est arête-reconstructible.

Démonstration:

Soit $G \in \mathcal{G}'$ et $(G_e)_{e \in E(G)}$ la liste des sous-graphes propres maxi-

maux de G. Comme G comporte une 5-clique alors il existe G_{e^o} comportant un sous-graphe induit isomorphe à:

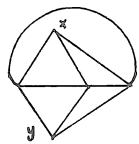


figure 1

Montrons que toute arête-reconstruction de G est isomorphe au graphe obtenu à partir de G_{e^o} en reliant les sommets x et y. En effet, s'il n'était pas ainsi, alors il existerait une arête-reconstruction de G avec un sous-graphe induit isomorphe au graphe de la figure 1 qui fait partie de la liste (1), ce qui est impossible d'après le lemme 1.

C.Q.F.D.

Si $G \in \mathcal{G}$ alors une clique d'ordre maximum de G est une 4-clique.

Lemme 4: $\mathcal{G}$ est arête-reconnaissable.

Démonstration:

Soit $G \in \mathcal{G}$ et H une arête-reconstruction de G. Il faut montrer que H est un graphe adjoint dont la clique maximum est d'ordre 4:

1) Si G est isomorphe a K_4, alors K_4 est la seule reconstruction de G car K_4 est arête-reconstructible (Harary (4)).

2) Si G n'est pas isomorphe à K_4:

- G comporte un sous-graphe sommet propre isomorphe à K_4. D'après le lemme 1, H comporte une 4-clique. Alors cette 4-clique est maximum car sinon H serait isomorphe a K_5 et donc G serait isomorphe à K_5 ce qui est impossible, ou H contiendrait proprement une 5-clique et donc G contiendrait une 5-clique d'après le lemme 1, ce qui est impossible.

- Pour montrer que H est un graphe adjoint, il suffit de montrer qu'il ne comporte pas de sous-graphe induit isomorphe à un des graphes de la liste (1): H ne peut pas etre isomorphe à un élément de cette liste car les graphes de cette liste comportant une 4-clique sont tous arête-reconstructibles (Harary (4)); de plus, aucun élément de la liste (1) ne peut pas etre isomorphe à un sous-graphe induit propre de H car dans ce cas G contiendrait aussi un tel sous-graphe d'après le lemme 1, ce qui est impossible. Donc, H

est un graphe adjoint.

C.Q.F.D.

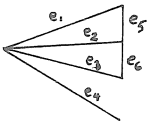
Proposition 2:

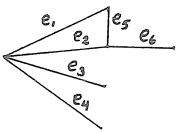
$\mathcal{G}$ est arête-recontructible.

Démonstration:

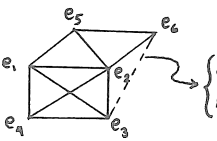
Soit $G \in \mathcal{G}$ et $(G_e)_{e \in E(G)}$ la liste des sous-graphes propres maximaux de G. Soit F un graphe sans sommets isolés avec L(F) isomorphe á G. On sait que $\Delta(F)=4$. Considérons un sommet de F de degré 4 :

1) Supposons que F contient un sous-graphe partiel isomorphe á:

F_1: 

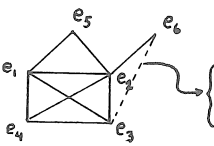
ou F_2: 

Alors, G contient le sous-graphe induit:



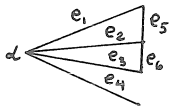
{avec cette arete pour F_1
sans cette arete pour F_2

Donc, il existe un G_{e_0} avec un sou-graphe induit isomorphe á:

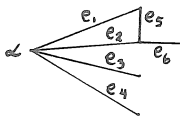
g_1: 

{avec cette arete pour F_1
sans cette arete pour F_2

Etant donné que toute arête-reconstruction de G contient un sous-graphe partiel isomorphe á g_1 alors tout graphe g' dont le graphe adjoint est une arête-reconstruction de G, contient le sous-graphe partiel:



pour F_1

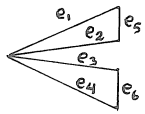


pour F_2

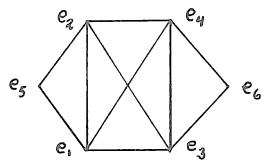
Dans g_1, e_6 est adjacent á e_2. Il ne peut pas etre adjacent á α dans g' car $\Delta(g')=4$. Par conséquent, e_6 est adjacent á e_5 dans g_1.

Ainsi, toute arête-reconstruction de G s'obtient à partir de G_{e^o} en joignant e_5 et e_6, deux sommets bien précis de G_{e^o}.

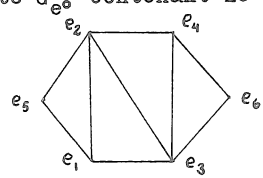
2) Supposons que F ne contient pas de sous-graphe isomorphe a' F_1 ou F_2 mais il contient un sous-graphe partiel isomorphe à:

F_3: 

G contient alors le sous-graphe induit:



Donc, il existe G_{e^o} contenant le sous-graphe induit:



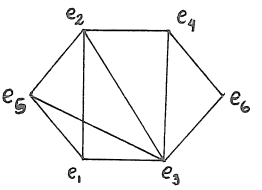
tel que e^o appartient à une 4-clique pour toute arête-reconstruction de G. (D'après le lemme 1, le nombre de 4-cliques contenant une arête spécifique est arête-reconstructible. Donc, on choisit un G_{e^o} dont ce nombre est positif).

Toute arête-reconstruction de G est un graphe adjoint. Par conséquent, d'après le lemme 2, e^o doit relier deux sommets du graphe précédent:

$e^o = [e_5,e_3]$, $[e_5,e_4]$, $[e_5,e_6]$, $[e_6,e_1]$, $[e_6,e_2]$ ou $[e_1,e_4]$.

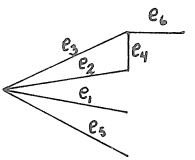
2.1.) Supposons que $e^o = [e_5,e_3]$:

On aurait une arête-reconstruction de G comportant le sous-graphe induit:



Cette arête-reconstruction ne peut pas etre isomorphe au graphe ci-dessus car ce graphe est arête-reconstructible (Harary (4)) et G serait aussi isomorphe á lui, ce qui est faux. Donc, le sous-graphe ci-dessus est un sous-graphe sommet propre de cette arête-reconstruction de G. D'aprés le lemme 1, toute arête-reconstruction de G contient un sous-graphe induit isomorphe au graphe ci-dessus et cela implique que tout graphe dont le graphe adjoint est une arête-reconstruction de G contient un sous-graphe isomorphe á:

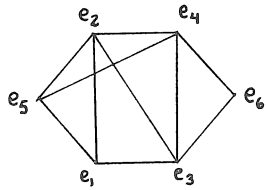
F_2:



en particulier F_2 ce qui est impossible d'aprés les hypothéses. Donc, $e^o \neq [e_3, e_5]$.

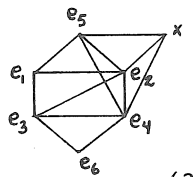
2.2.) Supposons que $e^o = [e_5, e_4]$:

On aurait une arête-reconstruction H de G avec le sous-graphe induit:



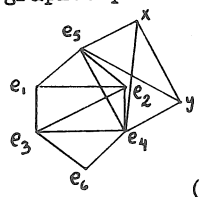
Comme on a supposé que e^o appartient á une 4-clique quelle que soit l'arête-reconstruction de G, cette 4-clique ne peut etre induite que par $\{e_5,e_4,e_2,x\}$ ou $\{e_5,e_4,x,y\}$, où x,y sont deux sommets différents de $e_1,e_2,\ \ldots\ ,e_6$.

H doit contenir donc un des sous-graphes partiels:



(2.2.1)

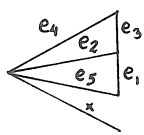
ou



(2.2.2.)

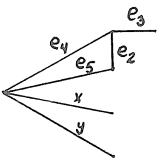
Si on a le cas (2.2.1.), ou bien H est isomorphe au graphe de la figure (2.2.1.), ou H contient un sous-graphe partiel arête propre isomorphe au graphe de la figure (2.2.1.). Dans le premier cas, G serait isomorphe á H car H est arête-reconstructible (Harary (4)). Dans le deuxiéme cas, d'aprés le lemme 1, G contiendrait un sous-graphe partiel isomorphe á celui de la figure (2.2.1.). Quoi qu'il en soit, le graphe F contiendrait un sous-graphe partiel isomorphe á:

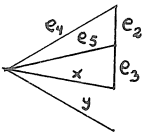
F_1:

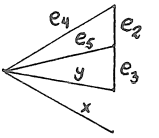


ce qui est impossible d'aprés les hypothéses.

Si on a le cas (2.2.2.), en raisonnant pareil au cas (2.2.1.), le graphe F contiendrait un sous-graphe partiel isomorphe á:

F_2: 

ou F_1: 

ou F_1: 

(Si e_3 n'est adjacent ni á x ni á y)

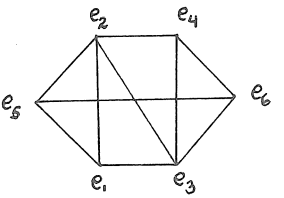
(Si x est adjacent á e_3)

(Si y est adjacent á e_3)

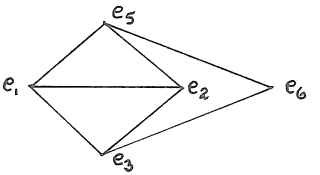
ce qui est impossible d'aprés les hypothéses. Donc, $e^\circ \neq [e_5, e_4]$.

2.3.) Supposons que $e^\circ = [e_5, e_6]$:

On aurait une arête-reconstruction de G contenant le sous-graphe induit:



Donc, on aurait une arête-reconstruction de G contenant le sous-graphe induit:



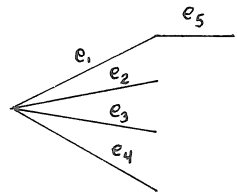
ce qui est impossible car ce graphe est dans la liste (1). Donc, $e^\circ \neq [e_5, e_6]$.

2.4.) Les cas $e^\circ = [e_6, e_2]$ et $e^\circ = [e_6, e_1]$ sont similaires aux cas (2.1) et (2.2) respectivement et on arrive á ce que $e^\circ \neq [e_6, e_2]$ et $e^\circ \neq [e_6, e_1]$.

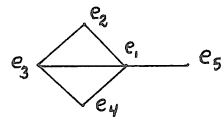
Finalement, seul le cas oú $e^\circ = [e_1, e_4]$ convient. Ainsi, toute arête-reconstruction de G est isomorphe au graphe obtenu á partir de G_{e° en reliant e_1 et e_4, deux sommets bien précis dans G_{e°.

3) Supposons que F ne contient pas de sous-graphe partiel isomorphe á F_1, F_2 ou F_3 mais il contient un sous-graphe partiel isomorphe á:

F_4:



Dans ce cas, il existe G_{e^o} contenant le sous-graphe induit:

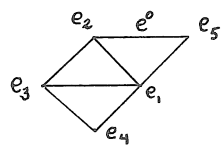


tel que e^o appartient á une 4-clique. (D'aprés le lemme 1, le nombre de 4-cliques contenant une arete spécifique est arête-reconstructible. Donc, on choisit un G_{e^o} dont ce nombre est positif).

Comme les sommets e_1, e_2, e_4, e_5 induisent $K_{1,3}$, on doit avoir $e^o = [e_2, e_5]$, $[e_4, e_5]$ ou $[e_2, e_4]$.

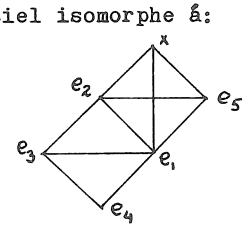
3.1) Supposons que $e^o = [e_2, e_5]$ (le cas $e^o = [e_4, e_5]$ est similaire)

On aurait une arête-reconstruction de G contenant le sous-graphe induit:

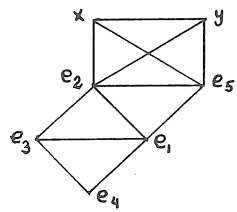


Comme e^o appartient á une 4-clique de cette arête-reconstruction, cette 4-clique serait induite par $\{e_1, e_2, e_5, x\}$ ou $\{e_2, e_5, x, y\}$, oú x,y sont deux sommets différents de $e_1, e_2, \ldots, e_5$.

On a donc une arête-reconstruction H de G contenant un sous-graphe partiel isomorphe á:

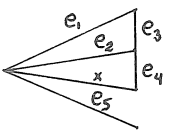


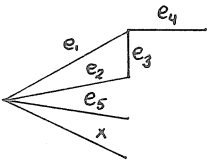
(3.1.1.)



(3.1.2)

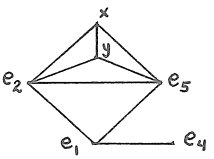
Dans le cas (3.1.1.), ou bien H est isomorphe au graphe de la figure (3.1.1.) ou H contient un sous-graphe partiel arête-propre isomorphe au graphe de la figure (3.1.1.). Dans le premier cas G serait isomorphe a H car H est arête-reconstructible (Harary (4)). Dans le deuxième cas, d'après le lemme 1, G contiendrait un sous-graphe partiel isomorphe á celui de la figure (3.1.1.). Quoi qu'il en soit, le graphe F contiendrait un sous-graphe partiel isomorphe á:

F_1: 

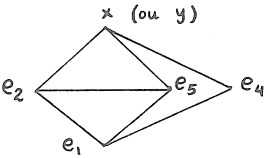
F_2: 

(Si e_4 est adjacent á x) (Si e_4 n'est pas adjacent á x)
ce qui est impossible d'après les hypothèses.

Dans le cas (3.1.2.), H contient un sous-graphe induit isomorphe á:



ou

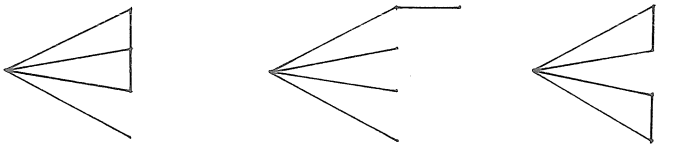


(Si e_4 nést adjacent ni á x ni á y) (Si e_4 est adjacent á x (ou y))

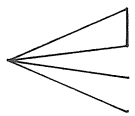
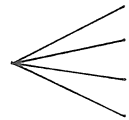
D'après le lemme 2, ceci est impossible.

Finalement on a $e^o = [e_2, e_4]$ et toute arête-reconstruction de G est isomorphe au graphe obtenu á partir de G_{e^o} en reliant e_2 et e_4, deux sommets bien précis de G_{e^o}.

4) Supposons que F ne tombe pas dans les cas (1), (2) ou (3). Donc, F ne contient pas de sous-graphe isomorphe aux graphes:



Pour tout sommet x de degré 4: $F\left[\{x\} \cup \Gamma(x)\right]$ induit une composante connexe isomorphe à:

F_5:  ou F_6: 

On sait que si F est non-connexe, alors G est non-connexe (car F ne possède pas de sommets isolés) et donc G est arête-reconstructible car G ne possède pas de sommets isolés. Si F est connexe, alors il est isomorphe à F_5 ou F_6 et on sait d'après Harary (4) que $L(F_5)$ et $L(F_6)$ sont arête-reconstructibles.

Finalement (1), (2), (3) et (4) impliquent que $\mathcal{G}$ est arête-reconstructible.

C.Q.F.D.

Théorême:

La classe des graphes adjoints des graphes simples de degré maximum ≥ 4 est arête-reconstructible.

BIBLIOGRAPHIE:

(1) Harary, Frank:
"On the reconstruction of a graph from a collection of subgraphs". Theory of graphs and its applications. (Proceedings of the Symposium held in Prague, 1964) édité par M. Fiedler. Prague. pp. 47-52.

(2) Bondy, J.A. ; Hemminger, R.L.:
"Graph Reconstruction - A Survey".
Journal of Graph Theory, vol. 1 (1977), 227-268.

(3) Greenwell, D.L.:
"Reconstructing Graphs". Proc. Amer. Math. Soc. 30 (1971), 431-433.

(4) Harary, F. ; Palmer, E.:
"A note on similar points and similar lines of a graph". Rev. Roumaine Math. Pures et Appli. 10 (1965), 1489-1492.

A. PEREDA B.*
R. CONTRERAS A.+

RESUMEN

Este trabajo trata de un algoritmo de ordenación, el cual se ha demostrado como bastante eficiente en aplicaciones en microcomputadores. Se desarrolla un análisis de su complejidad.

ABSTRACT

This work deals with a sorting algorithm, which has shown to be very efficient in microcomputers applications. A complexity analysis of the algorithm is developed.

* D.Sc. Computación (PUC/RJ, 1980); teoría de computación, estructuras de datos, ingeniería de software, Facultad de Ingeniería, Universidad de Concepción, Ciudad Universitaria, Concepción, Chile. Teléfono 24985 Anexo 2305.

+ M.Sc. Informática (PUC/RJ, 1983); ingeniería de software, estructuras de datos, lenguajes de programación, Facultad de Ingeniería, Universidad de Concepción, Ciudad Universitaria, Concepción, Chile. Teléfono 24985 Anexo 2305.

INTRODUCCION

El problema del costo del software, viene siendo atacado con diferentes enfoques, y variado éxito, por un sinnúmero de especialistas [2,8]. Creemos, sin embargo, que una de las tantas causas que inciden en este problema, es la utilización de técnicas (a veces) y algoritmos (puntualmente) inadecuados. Pese a existir una concepción filosófica marcadamente diferente pone aplicaciones enfocadas a microcomputadores y a computadores de gran porte, gran parte del software utilizado en los primeros es una simple adaptación de software desarrollado para máquinas grandes. Así, n o siempre el resultado de su utilización incide en un comportamiento eficiente.
En muchas aplicaciones, el tamaño de tablas a ordenar es pequeño (típicamente menos de 5.000 elementos), y es en este rango donde el algoritmo presentado ofrece ventajas respecto de los métodos tradicionales. Las pruebas iniciales demostraron la eficiencia de este algoritmo, lo que llevó a la realización del análisis que se mostrará a continuación.

EL ALGORITMO

Considerando como importantísimo el factor portabilidad, se eligió como lenguaje de implementación a PASCAL para la realización de las pruebas.
El algoritmo es el siguiente:

```
    procedure main
    procedure  asort (j,n);
1   begin
2      repeat  l ← J;
3      for  i ← j + l  to  n  do
4         if  T [ j ] > T [ i ]  then begin
5                                      l ← l + 1
6                                      T [ l ] ↔T[ i ]
7                                           end;
8      if  l ≠ j
9          then begin
10                    T [ l ] ↔ T[ j ];
11                    if l + 1 < n then  asort (l + 1, n);
12                    n ← l - 1
13                    end
14         else  j ← j + 1
15     until  j ≥  n
16    end
17  begin
18     asort (1 , n)       {n : tamaño de la tabla a ordenar}
19  end,   {main}
```

ANALISIS

Para llevar a cabo el análisis de complejidad del algoritmo asort, se determinará el número de veces que es ejecutado cada comando, al someter a ordenación una tabla de enteros de tamaño n.

Esto se consigue introduciendo contadores en la versión recursiva original, probando esta versión así modificada con las n! combinaciones posibles de los n primeros enteros positivos (TABLA 1).

n	n !	comando 3	comando 9	comando 14	Nº de llamadas recursivas
2	2	2	1	1	0
3	6	16	6	4	0
4	24	116	36	20	6
5	120	888	240	120	60
6	720	7416	1800	840	540
7	5040	67968	25120	6720	5040

TABLA 1

Por simple observación, se puede detectar que el número de veces que es ejecutado el comando 9 (rama then de if $l \neq j$) para el total de las n! tablas de tamaño n es $\frac{n!(n-1)}{2}$. De la misma manera se obtiene que la ejecución del comando 14 (else $j \leftarrow j+1$) se realiza $\frac{(n+1)!}{3!}$

Luego, el total de veces que se ejecuta el comando 8, al ordenar las n! tablas de tamaño n, es $\frac{n!\,(2n-1)}{3}$; esto da una media de ejecución de este comando de $\frac{(2n-1)}{3}$ veces por tabla.

En forma similar, se llega a que el número total de llamadas recursivas a asort es de $\frac{n!(n-3)}{4}$, para las n! tablas de tamaño n, correspondiendo una media de $\frac{(n-3)}{4}$ llamadas por tabla.

Si se desglosa el número de llamadas recursivas, según el tamaño de la tabla llamada, se obtiene el resultado mostrado en TABLA 2.

n	2	3	4	5	6	Total
4	6					6
5	36	24				60
6	252	168	120			540
7	2016	1344	960	720		5040

De esta tabla se puede concluir que el número de llamadas recursivas (NR) de las n! tablas posibles de tamaño n es:

$$n! * NR_n = (n+1)! \sum_{i=2}^{n-2} \frac{1}{(i+2)(i+3)}$$

En media, el número de llamadas recursivas realizadas durante la ordenación de una tabla de tamaño n es:

$$NR_n = (n+1) \sum_{i=2}^{n-2} \frac{1}{(i+2)(i+3)} \qquad n \geq 4$$

con $NR_3 = 0$, $NR_2 = 0$;

pero

$$\sum_{i=2}^{n-2} \frac{1}{(i+2)(i+3)} = \frac{(n-3)}{4(n+1)}$$

luego, este resultado coincide con el anterior obtenido a partir de la observación de valores totales.

Respecto al número de comparaciones de clave (if T [j] > T [i]...), se tiene el desglose mostrado en TABLA 3.

TABLA 3

tamaño / iteración n	1	2	3	4	5	6	Total
3	4	6					16
4	20	12	24				116
5	120	72	48	120			888
6	840	504	336	240	720		7416
7	6720	4032	2688	1920	1440	5040	67968

Esto da un total de comparaciones (C_n) para n! permutaciones de:

$$n!\ C_n = 2 \cdot (n+1)\ ! \sum_{i=2}^{n} \frac{(n-i)}{(n-i+2)\ (n-i+3)} + n!\ (n-1)$$

En media, el número de comparaciones realizadas al ordenar una tabla de tamaño n es

$$E\left[C_n\right] = 2\ (n+1) \sum_{i=2}^{n} \frac{(n-i)}{(n-i+2)\ (n-i+3)} + n - 1$$

Para determinar el valor de $E\left[C_n\right]$, se debe calcular el valor de la sumatoria

$$\sum_{i=2}^{n} \frac{(n-i)}{(n-i+2)\ (n-i+3)}$$

que se puede descomponer en:

$$= \sum_{i=2}^{n} \frac{n}{(n-i+2)\ (n-i+3)} - \sum_{i=2}^{n} \frac{i}{(n-i+2)\ (n-i+3)}$$

para la primera sumatoria

se tiene que

$$\sum_{i=2}^{n} \frac{n}{(n-i+2)\ (n-i+3)} = \frac{n\ (n-1)}{2\ (n+1)}$$

para calcular la segunda sumatoria hacemos $j = n-i+2$, lo que implica $i = n-j+2$, obteniendo:

$$\sum_{i=2}^{n} \frac{i}{(n-i+2)\ (n-i+3)} = \sum_{j=n}^{2} \frac{(n-j+2)}{j(j+1)}$$

luego

$$\sum_{i=2}^{n} \frac{(n-i)}{(n-i+2)\,(n-i+3)} = \frac{n\,(n-1)}{2\,(n+1)} - \sum_{j=n}^{2} \frac{(n-j+2)}{j\,(j+1)}$$

$$= \frac{n\,(n-1)}{2\,(n+1)} - (n+2) \sum_{j=n}^{2} \frac{1}{j\,(j+1)} + \sum_{j=n}^{2} \frac{1}{(j+1)}$$

$$= \frac{n\,(n-1)}{2\,(n+1)} - \frac{(n+2)\,(n-1)}{2\,(n+1)} + \sum_{i=3}^{n+1} \frac{1}{i}$$

$$= \frac{-2\,(n-1)}{2\,(n+1)} + \sum_{i=3}^{n+1} \frac{1}{i} = -\frac{(n-1)}{(n+1)} + \sum_{i=3}^{n+1} \frac{1}{i}$$

pero $\sum_{i=3}^{n+1} \frac{1}{i} = \gamma - 1.5 + \ln\,(n+1) + \frac{1}{2(n+1)} -$

$$\sum_{k=2}^{\infty} \frac{A_k}{(n+1)\,(n+2)\ldots(n+k+2)}$$

con $A_k = \frac{1}{k} \int_0^1 x(1-x)\,(2-x)\ldots(k-1-x)\,dx$

siendo γ la constante de Euler.

Luego:

$$\sum_{i=2}^{n} \frac{(n-i)}{(n-i+2)\,(n-i+3)} = \frac{(n-1)}{(n+1)} + \gamma - 1.5 + \ln\,(n+1) + \frac{1}{2(n+1)}$$

$$- \sum_{k=2}^{\infty} \frac{A_k}{(n+1)\,(n+2)\ldots(n+k+2)}$$

Entonces

$$E\,[\,C_n\,] = 2(n+1)\Big[-\frac{(n-1)}{(n+1)} + \gamma - 1.5 + \ln\,(n+1) + \frac{1}{2(n+1)} -$$

$$\sum_{k=2}^{\infty} \frac{A_k}{(n+1)\ldots(n+k+2)} \Big] + n - 1$$

$$E\,[\,C_n\,] = -\,2(n-1) + 2\,(\gamma - 1.5)(n+1) + 2(n+1)\ln\,(n+1) + 1 - 2(n+1)*$$

$$\sum_{k=2}^{\infty} \frac{A_k}{(n+1)\ldots(n+k+2)} + n - 1$$

y finalmente

$$E\,[\,C_n\,] = 2(n+1)\,\ln\,(n+1) + 2\gamma - 1 + (2\gamma-4)n - 2(n+1)\sum_{k=2}^{\infty}\frac{A_k}{(n+1)\ldots(n+k+2)}$$

El número de comparaciones, en el caso peor (número máximo de comparaciones para ordenar tabla de tamaño n) es $\frac{1}{2}\,n\,(n-1)$, y en el caso mejor (número mínimo de comparaciones para ordenar tabla de tamaño n) $2(n-2)$.

Es decir,

$$2(n-2) \leq C_n \leq \frac{1}{2}\,n\,(n-1)$$

En lo que se refiere al número de trueques realizados, se deben considerar dos factores, lo trueques $T\,[\,1\,] \leftrightarrow T\,[\,i\,]$ (comando 6) y los trueques $T\,[\,1\,] \leftrightarrow T\,[\,j\,]$ (comando 10).

En el segundo caso ($T\,[\,1\,] \leftrightarrow T\,[\,j\,]$), se sabe que, en media, para ordenar una tabla de tamaño n, se realizan $\frac{(n-1)}{2}$ trueques (número de veces que se ejecuta la rama then del if $1 \neq j$).

Para el caso del trueque $T\,[\,1\,] \leftrightarrow T\,[\,i\,]$, elnúmero de veces que este trueque es realizado se puede calcular, observando que el comando 6 se ejecuta (en media) la mitad de las veces que se realiza la comparación de claves ($T\,[\,j\,] > T\,[\,i\,]$), luego, este número es $(n+1)\,\ln\,(n+1) + \gamma - 0.5 + (\gamma-2)n - (n+1)\sum_{k=2}^{\infty}\frac{A_k}{(n+1)\ldots(n+k+2)}$

Entonces, el número total de trueques realizados(en media), al ordenar una tabla de tamaño n es la suma de este valor anterior con $\frac{(n-1)}{2}$; es decir

$$E\,[\,T_n\,] = (n+1)\,\ln\,(n+1) + \gamma - 1 + (\gamma - 1.5)n - (n+1)\sum_{k=2}^{\infty}\frac{A_k}{(n+1)\ldots(n+k+2)}$$

CONCLUSIONES

El algoritmo presentado, ofrece algunas ventajas, en cuanto al tiempo de ejecución, si se le compara, para tablas de menos de 5000 elementos, con una versión mejorada del QUICKSORT [7]. Se realizaron pruebas de comparación de tiempos, utilizando un computador VAX 11/780, mostrándose a continuación algunos de los resultados obtenidos (*)

(*) El tiempo de CPU está medido en milisegundos y las tablas fueron generadas utilizando una rutina random construida según la estrategia propuesta por Sedgewick [8]

n sort	500	1000	1500	2000	4000	4500
	80	180	270	430	960	1090
asort	60	180	270	370	1150	1120
	90	160	280	410	970	1120
	100	230	350	490	1120	1260
Quicksort	110	200	340	470	1000	980
	100	260	320	420	910	1160

TABLA 4

La complejidad del algoritmo propuesto, O(n log n), para el comportamiento promedio, y los tiempos mostrados en la tabla anterior además de su simplicidad de implementación, lo hacen apropiado para su utilización en microcomputadores.

BIBLIOGRAFIA

[1] Gonnet, G.H. "Handbook of Algorithms and Data Structures" Addison-Wesley P.C. (1984)

[2] Horowitz, E., Sahni, S. "Fundamentals of Computer Alrorithms" Computer Science Press. (1978)

[3] Knuth, D. "The Art of Computer Programming" Vol 1. 2nd Ed. Addison-Wesley P.C. (1975)

[4] Knuth, D. "The Art of Computer Programming" Vol 3. Addison-Wesley P.C. (1973)

[5] Larsen, H.D. "Rinehart Mathematical Tables, formulas and curves" Holt, Rinehart and Winston. (1960)

[6] Ryshik, Gradstein "Tables" VEB Deutscher Verlag der Wissenschaften, DDR. (1963)

[7] Sedgewik, R. "Quicksort" Computer Science Dept. Stanford U. Technical Report STAN-CS-75-492. (1975)

[8] Sedgewik, R. "Algorithms" Addison-Wesley P.C. (1983)

Jussara Maria Marins*

SUMÁRIO

Este trabalho tem por finalidade apresentar as noções básicas dos algoritmos de quadratura adaptativas. Inicialmente mostra uma visão geral dos componentes das rotinas adaptativas. A seguir, dá uma fundamentação teórica simples para uma abordagem geral. A implementação da rotina didática é feita pela aplicação de regra de Simpson num procedimento em PASCAL. Finalmente 3 exemplos são apresentados para mostrar o comportamento típico da rotina.

ABSTRACT

This work is intend to present the basic notions of adaptative quadrature algorithms. Inicially,it shows a general view of the components of adaptative routines. Folowing this, it gives a simple the oretical foundation for a general approach. The implementation of the didactic routine is done by application of Simpson's rule, in a PASCAL procedure. Finally 3 exemples are presented to show the tipical behavior of the routine.

* Professora Assistente do Instituto Tecnológico de Aeronáutica (ITA); quadratura, integração numérica, programa para integração numérica.

INTRODUÇÃO (1)

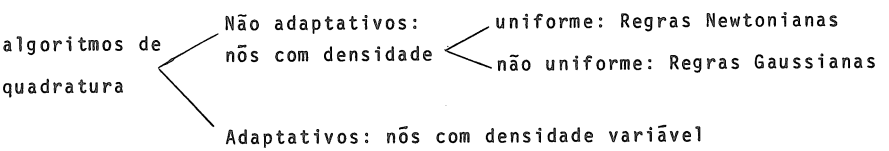
O problema da quadratura é determinar o valor de

$$(1) \quad I(f) = \int_a^b f(x)\,dx$$

O objetivo da quadratura numérica é calcular o valor de (1) usando exclusivamente operações aritméticas. Para estimar o valor de (1) vamos aproximá-la por uma combinação de valores da função em certos pontos x_i, $-\infty \leqslant a \leqslant x_i \leqslant b \leqslant +\infty$, chamados de nós e certos valores w_i, que constituem pesos, ou seja:

$$(2) \quad \int_a^b f(x)\,dx \cong w_1 f(x_1) + w_2 f(x_2) + \ldots + w_n f(x_n) + w_{n+1} f(x_{n+1})$$

De acordo com os valores dos pesos e com a escolha dos nós temos, no lado direito de (2) o que costumamos chamar de Regra de Integração. As diferentes filosofias surgem na maneira de fazer a escolha dos nós, que se refletem no tipo de algoritmo usado. Numa abordagem simples os algoritmos podem ser classificados como mostra o seguinte esquema:



O objetivo das quadraturas adaptativas é calcular o valor aproximado de (1) variando o tamanho dos subintervalos, de modo que, nos trechos onde a curva é mais abrupta, o número de subintervalos é maior, enquanto que tal número é menor onde a curva é mais suave. Para fazer o cálculo do valor aproximado de (1) usa-se uma ou mais regras básicas de quadraturas Newtonianas ou gaussianas.

Os algoritmos adaptativos são geralmente complicados pois podem fazer uma análise complexa do comportamento do integrando, dependendo dos limites de erro parciais que apresentem em cada subintervalo.

O objetivo deste trabalho é apresentar uma rotina adaptativa simples que possa ser ensinada num curso básico de cálculo numérico para graduação. Contudo, apesar da simplicidade esta rotina permite que faça novas estruturas ampliando os resultados.

Na seção 2 veremos com mais detalhes os diferentes modos de implementar rotinas adaptativas e um esquema geral para quadraturas adaptativas. Na seção seguinte veremos uma fundamentação simples sobre o estudo de um critério de convergência baseado em Forsythe [1]. Na seção 4 veremos os deta

lhes da implementação da rotina adaptativa didática e finalmente na seção 5 veremos exemplos mostrando o desempenho da rotina.

FILOSOFIAS ADAPTATIVAS (2)

Conforme vimos na introdução, uma rotina adaptativa é um algoritmo que usa uma ou mais regras básicas de quadratura e que conforme for o comportamento de f, escolhe dinamicamente o tamanho dos subintervalos e os pesos w_i de (2) de modo que o resultado computado encontre um ou mais critérios de exatidão pré-determinados. Podemos mostrar um exemplo gráfico simples na figura 1, que mostra a escolha dos tamanho dos subintervalos.

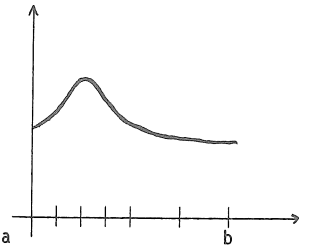


Figura 1

Diferentes tamanhos podem ser usados ao longo do intervalo de integração [a,b] com subintervalos relativamente grandes onde o integrando é suave e de variação lenta e pequenos subintervalos em regiões onde a integração fica mais difícil. Observando a figura 1 é fácil verificar em que região a curva é mais abrupta, mas como verificar isto automaticamente? Pelo exemplo, também observamos na figura 1 que se dividirmos o intervalo [a,b] em subintervalos de tamanho fixo e de tamanho pequeno teriamos um resultado que também prescreveria o critério de exatidão mas a um custo maior. Em tais rotinas de quadratura uma medida de eficiência do algoritmo é o número de avaliações de f que é feito para se atingir a exatidão exigida, o que está bastante ligado com o custo final do processamento. De um modo geral, se uma função é integrada por duas rotinas diferentes com a mesma exatidão a rotina que requer menor número de avaliações da função é vista como a mais eficiente para o problema particular. Usualmente o usuário de rotinas adaptativas ou automáticas não quer pensar qual a melhor regra para usar e qual o melhor número de subdivisões que deverá ser feita para que se tenha uma boa exatidão. Ele quer apenas fornecer a rotina para calcular $f(x)$ ou os pontos dados (x_i, y_i); os limites de integração, uma tolerância ε para o limite de erro e um limite de segurança L para o número máximo de

avaliações de f. Sendo I(f) o valor exato de (1) e R(f) um resultado aproximado de (1) o usuário deseja que

$$(3a)\quad |I(f) - R(f)| \leq \varepsilon_1 \quad \text{ou} \quad (3b)\quad \frac{|I-R(f)|}{|R(f)|} \leq \varepsilon_2$$

onde ε_1 é a tolerância para o erro absoluto e ε_2 é a tolerância para o erro relativo. Em qualquer caso, o programa fornece o erro "teórico"que foi obtido e que é usado com controle da saída.

A rotina pode decidir que a exatidão prescrita não é atingível dentro do limite L, faz o melhor que pode e retorna com um resultado confiável ou um resultado parcial com alguma mensagem. É sempre possível achar integrandos que "enganam" a rotina produzindo um resultado errado, mas para as boas rotinas a classe de tais exemplos deve ser tão pequena quanto possível sem complicar em demasia a lógica da rotina fazendo-a ineficiente em casos razoáveis.

Para entendermos melhor as diferentes filosofias adaptativas precisamos detalhar os componentes básicos de uma rotina adaptativa segundo RICE [2]. São eles **processador de intervalos**; a **coleção de intervalos**;o gerenciador da **coleção de intervalos** e o **controlador de algoritmo**.

O processador de intervalos faz os cálculos necessários para prover as aproximações para o valor da integral através das regras de integração e calcula os limites de erro associados a estes intervalos. Aqui podemos ter:

- uma única regra aplicada a todo intervalo e a cada metade
- regras independentes da mesma ordem
- regras de ordem diferentes, e outras

O gerenciador da coleção de intervalos cria e mantém a coleção de intervalos numa determinada estrutura que pode ser uma pilha; uma lista ordenada (neste caso os intervalos são ordenados de acordo com os valores das estimativas de erro); uma fila; uma fila onde intervalos com estimativas de erro são descartados ou caixas.

O controlador do algoritmo pode ter dois componentes distintos: o **estimador do limite** de erro que é envolvido na decisão de terminar o algoritmo e uma **coleção de procedimentos** para detectar tipos especiais de comportamento como singularidades, descontinuidades e problemas com erro de arredondamento. Variando os elementos de cada componente podemos ter um número muito grande de algoritmos diferentes. É bastante difícil identificar dentre todos estes algoritmos quais são os realmente superiores. Para esta análise utilizam-se 5 ferramentas que iremos apenas citá-las.

Análise de erro e convergência, estudo da complexidade computacional; análise dos componentes dos algoritmos adaptativos; estudos da população de integrandos e estudos experimentais. Tais análises fogem ao objetivo des-

te trabalho. Na próxima seção veremos uma fundamentação simples sobre as rotinas adaptativas.

FUNDAMENTAÇÃO (3)

A base da idéia adaptativa é subdividir o intervalo de integração num número maior de subintervalos onde a variação de f é mais acentuada. Para que se faça isso automaticamente tomamos duas aproximações da integral num subintervalo e as comparamos para obter uma estimativa da sua exatidão. Se esta é aceitável uma das duas é tomada como valor da integral sobre o intervalo. Se a exatidão não é aceitável o subintervalo é dividido em duas ou mais partes e o processo é repetido nos subintervalos menores. Para reduzir o número de avaliações da função usaremos a bissecção do intervalo de modo que a segunda regra pode ser calculada usando os pontos utilizados pela primeira regra. Vamos aplicar uma única regra aplicada em todo subintervalo que vamos denotá-lo por $[x_i, x_{i+1}]$. A numeração é arranjada de modo que $x_1 = a$ e a rotina determina n tal que, no final do processo, $x_{n+1} = b$. Seja h_i o tamanho de cada intervalo, ou seja $h_i = x_{i+1} - x_i$. Vamos denotar por P_i a aplicação da regra no subintervalo $[x_i, x_{i+1}]$ e Q_i a soma da aplicação mesma em cada metade deste subintervalo. Tanto P_i como Q_i são aproximações para

$$(4) \qquad I_i = \int_{x_i}^{x_{i+1}} f(x)\,dx$$

Vamos aplicar regras newtonianas para o cálculo de P_i de modo que P_i dá a resposta exata se o integrado é um polinômio de grau p-1, ou seja, que $f^{(p)}(x)$ é identicamente nula. Pode ser mostrado que

$$(5) \qquad I_i - P_i = c h_i^{p+1} f^{(p)}\left(x_i + \frac{h_i}{2}\right) + \ldots$$

Como Q_i é a soma do dois P_i's de subintervalos de comprimento $h_i/2$ segue que:

$$(6) \qquad I_i - Q_i = c\left(\frac{h_i}{2}\right)^{p+1}\left[f^{(p)}\left(x_i + \frac{h_i}{4}\right) + f^{(p)}\left(x_i + \frac{3h_i}{4}\right)\right] + \ldots$$

desde que

$$(7) \qquad f^{(p)}\left(x_i + \frac{h_i}{4}\right) + f^{(p)}\left(x_i + \frac{3h_i}{4}\right) = 2f^{(p)}\left(x_i + \frac{h_i}{2}\right) + \ldots$$

e reescrevendo (6), usando (5) e (7) temos

$$(8) \qquad I_i - Q_i = \frac{1}{2^p}\,(I_i - P_i),$$

indicando que, bisseccionando o intervalo o erro diminui por um fator de 2^p. Resolvendo para a incógnita I_i e rearranjando temos:

$$(9) \qquad Q_i - I_i = \frac{1}{2^p-1}\,(P_i - Q_i),$$

em outras palavras, o erro em Q_i é aproximadamente $\frac{1}{2^p-1}$ vezes a diferença entre duas aproximações.
Portanto se queremos que $I(f) - R(f)<\varepsilon$ e se tomamos

$$(10) \qquad R(f) = \sum_{i=1}^{n} Q_i$$

teremos que a subdivisão de um subintervalo em outros subintervalos menores deve ser feita até que se atinja.

$$(11) \qquad |p_i - Q_i| < \frac{(2^p-1)^{h_i}\,\varepsilon}{b-a} = F(\varepsilon, p, a, b, h)$$

onde ε é o critério de exatidão fornecido. Mostramos agora que se (11) é válida em todos subintervalos então (3a) é satisfeita. Para isso vejamos.

$$|R(f) - I(f)| = \left|\sum_{i=1}^{n} (Q_i - I_i)\right|$$

$$\leq \sum_{i=1}^{n} |Q_i - I_i|$$

que por (9) ignorando os termos de mais alta ordem temos

$$|R(f) - F(f)| \leq \frac{1}{2^p-1} \sum_{i=1}^{n} |P_i - Q_i|$$

Como (11) é satisfeita em cada subintervalo temos

$$(12) \qquad |R(f) - I(f)| \leq \frac{1}{2^p-1} \cdot \frac{(2^p-1)}{b-a}\,\varepsilon \cdot \sum_{i=1}^{n} h_i \leq \varepsilon.$$

Esta análise requer que $f^{(p)}(x)$ seja contínua e que o erro de truncamento seja proporcional a $h_i^{(p+1)}\, f^{(p)}(x)$. Esta condição nem sempre será satisfeita para todo integrando, contudo conforme for implementado o controlador do algoritmo poderemos ter um resultado confiável atingindo a exatidão desejada.

IMPLEMENTAÇÃO (4)

Como esta é uma rotina didática e não competitiva com as demais existentes usaremos o que há de mais simples em cada um dos componentes de um al

goritmo adaptativo citado em 2. O processador de intervalos utilizará a regra de Simpson simples aplicada em todo intervalo $[x_i, x_{i+1}]$ e a mesma regra composta aplicada em cada metade do subintervalo.
Teremos então

$$(13)\quad P_i = \frac{h_i}{6}\left[f(x_i) + 4f(x_i + \frac{h}{2}) + f(x_i+h)\right]$$

e

$$(14)\quad Q_i = \frac{h_i}{12}\left[f(x_i) + 4f(x_i + \frac{h_i}{4}) + 2\, f(x_i + \frac{h}{2}) + 4f(x_i + \frac{3h_i}{4}) + f(x_i + h_i)\right]$$

que pode ser visualizado na seguinte figura 2.

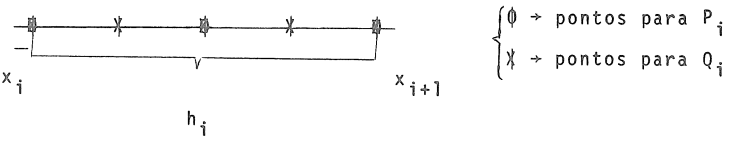


Figura 2

Como para a regra de Simpson p = 4 então o estimados de limite de erro será dado por (11) ou seja, cada intervalo será bisseccionado enquanto

$$(15)\quad |P_i - Q_i| > \frac{16\, h_i\, \varepsilon}{(b-a)}$$

Nesta rotina não forneceremos um limite de erro para cada subintervalo mas ele pode ser facilmente obtido. Esta rotina não tem nenhum mecanismo especial para detectar singularidades ou erros de arredondamentos. Toda vez que (11) não é satisfeita então o subintervalo, assim como as informações já obtidas são colocados numa coleção de intervalos que obedece a uma estrutura de pilha. Esta estrutura é finita e pode armazenar um número pré-determinado de subintervalos que não atingiram (11). Então é necessário um controle no número de subdivisões feitas para não causar overflow na pilha. Vamos especificar que o número máximo de subdivisões permitidas é N. Tal número deve ser escolhido em função da precisão que a máquina onde será implementado o algoritmo pode oferecer. Outro fator a ser considerado é que embora tal limite possa não ser atingido é preciso limitar o número de avaliações da função que será indicado por L. Vejamos agora o esquema simplificado para uma rotina adaptativa.

1. entrada {f, a, b N, L, ε}
2. inicializa: $h_i \leftarrow b-a$; $e \leftarrow a$; $x_o \leftarrow a$; $R \leftarrow 0$; $Q \leftarrow 0$.
3. calcular P_i e Q_i
4. se $|P_i - Q_i| < F(\varepsilon, p, a, b)$

então
- $Q \leftarrow Q + Q_i$
- se a pilha está vazia
  - então $R \leftarrow Q$ e pára
  - senão desempilha e volta a 2

Senão
- empilha $\left[e + \frac{h_i}{2}, e + h_i\right]$
- $h_i \leftarrow h_i/2$
- volta a 2.

A rotina INARS (Integração Numérica Adaptativa pela Regra de Simpson) implementada em PASCAL no IBM-4341 é dada no final deste trabalho.

DESEMPENHO (5)

Vejamos 3 exemplos que mostram o desempenho adaptativo da rotina INARS veremos exemplos com um número pequeno de subdivisão para não tornar cansativo os exemplos.

Exemplo 1 $\int_0^1 \frac{1}{1+x}\,dx$ cujo gráfico é dado na figura 3

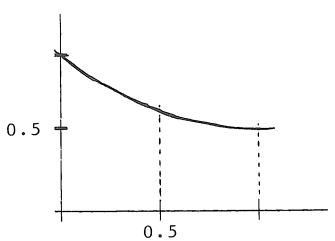


Figura 3

e o valor exato é dado por $\ell_n 2 - \ell_n 1 = 0.693\ 147\ 180\ 6$

Executando passo a passo a rotina temos:

h = 1.0, N = 0, E = 0.0, F.(ε, p, a, b, h) = 15 ε h_i
P = 6.944 444 4E-1

Q = 6.932 539 6 E-1
|P-Q| = 0.001 190 48 $\not<$ 15 * 1.0 E-1 = 0.000 15
empilha: [0.5, 1.0]
h = 0.5, E = 0.0 P = 4 055 555 5 E-1 Q = 4.054 713 8 E-1
|P-Q| = 0.000 084 17 $\not<$ 15 * 0.5 * 1.0 E-1 = 0.000 075
empilha [0.25, 0.5]
h = 0.25, E=0.0, P = 2.231 481 4 E-1, Q = 2.231 438 4 E-1
|P-Q| = 0.000 004 34 < 15 * 0.25 * 1.0 E-5 = 0.000 037 5
R = 2.231 438 4 E-1
desempilha [0.25, 0.5]
h = 0.25, E = 0.25 P = 1.823 232 3 E-1
Q = 1.823 216 6 E-1
|P-Q| = 0.000 001 57 < 15 * 0.25 + ε = 0.000 037 5
R = 0.405 465 5
desempilha:
h = 0.5 e = 0.5 P = 2.876 984 12 E-1
Q = 2.876 838 5 E-1
|P-Q| = 0.000 015 162 < 15 * 0.5 * ε = 0.000 075
R = 0.693 148 65

E dai temos que |R(f) - I(f)| = 1.46 E-6 < 1.0 E-5.
Tal resultado foi obtido com 13 avaliações do integrando f, conforme está representado graficamente na figura 4, o que está de acordo con o esperado.

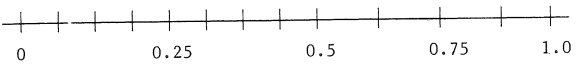
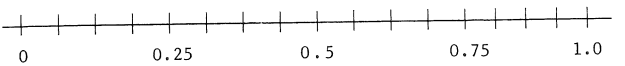


Figura 4

Para obter a mesma exatidão com uma rotina não-adaptativa precisamos de 17 avaliações de f.

Exemplo 2 $\int_0^0 \sin \Pi \, dx$

Neste caso não faremos os cálculos detalhadamente mostrando apenas na figura 5 a distribuição dos pontos no intervalo [0,1], que é uniforme pois f é suave em toda extensão de [0.1].



O resultado exato é 2/Π = 0.636 619 772 4 enquanto que o resultado aproximado é 0.636 625 064.

Exemplo 3 $\int_0^1 \sqrt{x}\ dx$

Neste caso, temos que f não tem primeira derivada limitada, no entanto, o resultado aproximado obtido é 0.666 663 238 com 61 avaliações do integrando f pois esta função tem um crescimento muito rápido em torno do ponto zero onde f' não é definida. O menor valor de h_i foi de 6.103 52 E-5.
Neste caso com uma quadratura fixa precisariamos de um número de avaliações de f muito maior.

BIBLIOGRAFIA

|1| FORSYTHE, G.E.; MALCOLM, M.A. & MOLER C.B. Computer methods for mathematical computations. Prentice-Hall, INC, 1977.

|2| RICE, J.A. Metalgorithm for adaptative quadrature. J.ACM 22, 1(Jan 1975) 61-82.

|3| LYNESS, J.N. Notes on the adaptative Simpson quadrature routine. J ACM 16, (July 1969), 483-495.

```
PROCEDURE INARS(FUNCTION F(P REAL) REAL
                        A,B,EPS   REAL
                VAR     RES       REAL
                        NF INTEGER)
VAR E, PHI,H,X , P , Q   REAL
    N,NMAX,NFMAX,I,K     INTEGER
    CONTINUA             BOOLEAN
    XI              ARRAY (.1..5.) OF REAL
    XAUX            ARRAY (.1..20.) OF REAL
    FAUX                 ARRAY (.1..42.) OF REAL
(*                      *
 *   INICIALIZACAO      *
 *                      *)
BEGIN
WRITELN('CALCULOS COM EPSILON =',EPS)
E =A
RES =0.0
NF =5
N =0
H =B-A
NMAX =20
NFMAX =1000
PHI =15.0*EPS/(B-A)
CONTINUA =TRUE
FOR I =1 TO 5 DO
   BEGIN
   X =E+(I-1)*H/4
   XI(.I.) =F(X)
   END
(*
 *    LACO PRINCIPAL *
 *                   *)
WHILE CONTINUA DO
   BEGIN
   (*                                *
    *    CALCULO DAS APROXIMACOES    *
    *                                *)
   P =(H/6)*(XI(.1.) + 4 * XI(.3.) + XI(.5.))
   Q =(H/12)*(XI(.1.) + 4 * XI(.2.) + 2 * XI(.3.) + 4 * XI(.4.) +
              XI(.5.))
   WRITELN(' H= ',H 15 10,'  E= ',E 15 10, ' P= ',P,' Q= ',Q)
   (*                                                 *
    *  VERIFICA SE SUBINTERVALO CONVERGIU              *
    *                                                 *)
    IF ABS(P-Q)   PHI*H THEN
       BEGIN
       (* ACUMULA RESULTADOS PARCIAIS *)
       RES =RES+Q
       WRITELN(' NIVEL =',K 5,'RES.PARC = ',RES)
       (**
        *  VERIFICA SE PILHA ESTA VAZIA  *
        *                                *)
       IF N  = 0 THEN
                 (*                          *
                  *  PILHA ESTA VAZIA        *
                  *                          *)
                 BEGIN
                 (*                  *
                  *  FIM NORMAL      *
                  *                  *)
                 WRITELN('RESULTADO= ',RES,' COM ',NF 5,' AVALIACOES ',
                         'DE F(X) ')
                 CONTINUA =FALSE
                 END
                 ELSE
                 BEGIN
                 (*                              *
                  * PILHA NAO ESTA VAZIA         *
                  *                              *
                  *  DESEMPILHA                  *
                  *                              *)
                 WRITELN('DESEMPILHA ')
                 WRITELN('              XINIC=',E 15 10,' XFIN=',(E+H)
                 15 10,' H=',H 15 10)
                 E =E+H
                 H =XAUX(.K.)-E
                 XI(.1.) =XI(.5.)
                 XI(.3.) =FAUX(.N.)
                 XI(.5.) =FAUX(.N-1.)
                 N =N-2
                 K =N DIV 2
```

```
                (**
                 * VERIFICA SE ATINGIU NUMERO MAXIMO DE AVALIACOES  *
                 *                                                  *)
                IF NF NFMAX THEN
                        BEGIN
                        WRITELN('EXCEDEU ',NFMAX 5,'AVALIACOES DE',
                        ' DE F(X) ')
                        CONTINUA =FALSE
                        END
                        ELSE
                        BEGIN
                        (*                          *
                         *     REFAZ CALCULOS       *
                         *                          *)
                        NF =NF+2
                        X =E+H/4
                        XI(.2.) =F(X)
                        X =E+3*H/4
                        XI(.4.) =F(X)
                        END
                END
        END
        ELSE
        BEGIN
        (*                                                          *
         *     SUBINTERVALO NAO CONVERGIU                           *
         *                                                          *
         *  VERIFICA SE ATINGIU NUMERO MAXIMO DE SUBDIVISOES        *
         *                                                          *)
        IF (N DIV 2) NMAX THEN
                BEGIN
                WRITELN('EXCEDEU NUMERO MAXIMO DE SUBDIVISOES')
                CONTINUA =FALSE
                END
                ELSE
                BEGIN
                (*                                                  *
                 * NAO ATINGIU NUMERO MAXIMO DE SUBDIVISOES         *
                 *                                                  *
                 * EMPILHA                                          *
                 *                                                  *)
                WRITELN('EMPILHA')
                N =N+2
                K =N DIV 2
                FAUX(.N.) =XI(.4.)
                FAUX(.N-1.) =XI(.5.)
                XAUX(.K.) =E+H
                H =H/2
                (*                  *
                 *   TRANSLADA      *
                 *                  *)
                XI(.5.) =XI(.3.)
                XI(.?.) =XI(.2.)
                (*
                 * VERIFICA SE EXCEDEU NUMERO MAXIMO DE AVALIACOES *
                 *)
                IF NF NFMAX THEN
                        BEGIN
                        WRITELN('EXCEDEU ',NFMAX 5,'AVALIACOES',
                        'DE F(X) ')
                        CONTINUA =FALSE
                        END
                        ELSE
                        BEGIN
                        (*                          *
                         *    REFAZ CALCULOS        *
                         *                          *)
                        NF =NF+2
                        X =E+H/4
                        XI(.2.) =F(X)
                        X =E+3*H/4
                        XI(.4.) =F(X)
                        END

                END
    END
  END (* WHILE *)
END (* IMAFS *)
```

METODOLOGIA PARA O DESENVOLVIMENTO DE UM SISTEMA DE AQUISIÇÃO DE DADOS E DE SUPERVISÃO

J.M. FARINES*, R.S. DE LEMOS**, L.F. BORGES DE ANDRADE***

SUMÁRIO

Neste artigo, é apresentada uma metodologia para o desenvolvimento da parte material e de programação de um sistema de aquisição de dados e supervisão, tomando como exemplo o desenvolvimento de um sistema de aquisição de dados para um gaseificador de madeira de uma usina piloto para produção de metanol a partir da madeira. As várias técnicas de análise dos requisitos, projeto e testes utilizadas durante este desenvolvimento serão mostradas e discutidas.

ABSTRACT

In this paper, is presented a metodology for development of hardware and software for data aquisition and supervision system. The various phases are described, based to an example of aquisition data system development for a wood gasification process to produce methanol. For these phases, corresponding techniques are shown and examined.

* Engenheiro (INPT-1972); Doutor-Engenheiro (INPT-1979); controle de processos, sistemas de aquisição e supervisão, software em tempo real, C.A.D. para sistemas de controle - Laboratório de Controle e Microinformática - Depto. Engenharia Elétrica - CP 476 - UFSC 88000 - Florianópolis - SC.

** Engenheiro (UFSC-1983); controle de processos, sistemas de aquisição e supervisão, hardware - Laboratório de Controle e Microinformática - Depto. Engenharia Elétrica - CP 476 - UFSC - 88000 - Florianópolis - SC.

*** Engenheiro (UFSC-1983); controle de processos, sistemas de aquisição e supervisão; Laboratório de Controle e Microinformática - Depto. Engenharia Elétrica - CP 476 - UFSC - 88000 - Florianópolis - SC.

INTRODUÇÃO

A diminuição dos custos da microinformática e das tecnologias adjacentes (em particular, para a transmissão de dados), a necessidade de uma melhoria na qualidade dos produtos e de uma maior economia de energia e a exigência de uma flexibilidade e de uma confiabilidade cada vez maiores levaram a uma crescente automatização dos processos industriais. Entre as atividades industriais automatizadas de maior importância hoje, se encontram a monitoração, a supervisão e o controle das plantas ou de parte delas.

O processo de monitoração, supervisão e controle segue, na maior parte dos casos, a filosofia da descentralização ou da distribuição das funções. Em consequência, se torna necessária a existência de unidades de aquisição de dados, supervisão e controle repartidas em diferentes pontos de uma planta ou de um sistema e em comunicação entre si. Outrossim, algumas destas unidades podem, quando utilizadas autonomamente, desempenhar também um papel importante na fase de análise, projeto e teste de uma planta piloto ou parte desta.

Neste trabalho, é apresentada uma metodologia para o desenvolvimento da parte material (hardware) e de programação (software) de um sistema de aquisição de dados e supervisão que deve funcionar autonomamente sobre uma planta piloto; entretanto, procurar-se-á, no decorrer do desenvolvimento deste sistema, levar em conta a sua futura introdução num sistema distribuído.

O desenvolvimento de um sistema de aquisição de dados e supervisão para um gaseificador de madeira instalado na usina piloto da CESP em Rio Claro servirá para exemplificar os vários aspectos desta metodologia. A figura 1 representa o diagrama-bloco deste gaseificador que permite a transformação da madeira num gás de síntese a ser utilizado posteriormente na produção de metanol. A gaseificação se faz pela passagem de uma corrente elevada entre os eletrodos e o processo é controlado pela injeção de vapor na zona de reação. Estudos de rendimento e análise do comportamento do gaseificador estão sendo realizados para melhor conhecer o processo e otimizar o projeto.

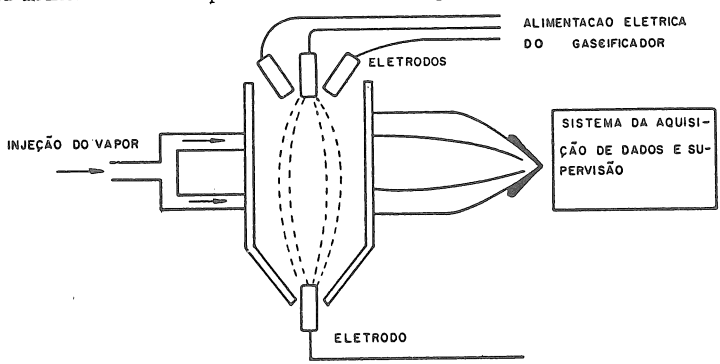


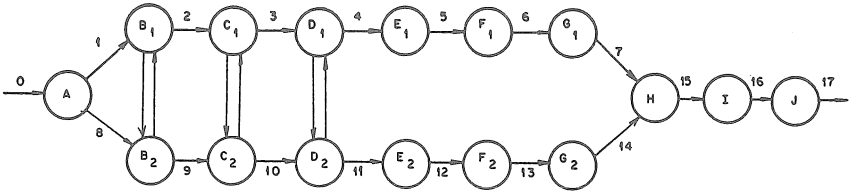
Figura 1: Diagrama-bloco da planta piloto

METODOLOGIA GERAL DE DESENVOLVIMENTO DO SISTEMA

A importância da escolha de uma metodologia para o desenvolvimento das partes material e de programação de um sistema se deve principalmente a melhor qualidade do produto final e da sua documentação, a racionalização das várias etapas do desenvolvimento e do gerenciamento destas e a diminuição dos custos de desenvolvimento e de manutenção decorrentes.

Na figura 2, são apresentadas as várias fases do desenvolvimento de um sistema e o seu encadeamento. As fases principais são as seguintes:

- Planejamento e especificações divididas em:
  - definição do sistema (A)
  - planejamento do hardware (B1) e do software (B2)
  - análise de requisitos do hardware (C1) e do software (C2)
- Desenvolvimento propriamente dito com as seguintes etapas:
  - projeto preliminar e detalhado do hardware (D1) e do software (D2)
  - implementação: montagem da parte material (E1) e codificação (E2)
  - teste de unidade e de integração do hardware (F1) e do software (F2)
  - teste de validação do hardware (G1) e do software (G2)
- Instalação do sistema:
  - "assemblagem" final do hardware e do software (H)
  - teste do sistema total (I)
  - instalação e teste de aceitação no local (J)
- Manutenção



pedido do usuário: 0
especificações informais do hardware (1) e do software (8)
plano de hardware (2) e de software (9)
especificações funcionais formais do hardware (3) e do software (10)
diagrama-bloco da parte material (4); módulos descritos em pseudo-código (11)
placas montadas com esquemas (5); programa codificado com documentação (12)
placas testadas (6); módulos de programa testados e integrados (13)
hardware validado (7) e software validado (14)
hardware e software "assemblados" (15)
sistema pronto para entrega (16)
sistema instalado e aceito (17)

Figura 2: Fases do desenvolvimento

Cada uma destas fases necessita das informações obtidas nas fases anteriores e transmitidas na forma de documentação. O desenvolvimento do sistema segue dois caminhos principais: o do hardware e o do software muitas vezes realizados independentemente. Entretanto, no caso de sistemas utilizando microcomputadores, é necessário a comunicação entre estes dois caminhos, principalmente nas etapas de planejamento e análise de requisitos; isto permitirá diminuir a complexidade durante a fase de "assemblagem" final e evitar erros irreversíveis. As informações vindo de um caminho funcionam como restrições para o outro no decorrer de todo o desenvolvimento do sistema.

Revisões e procura de erro ("walkthrough") são realizadas no final de cada fase e em especial na fase de planejamento e especificações, com a participação do usuário.

DEFINIÇÃO DO SISTEMA

Nesta fase, foram estabelecidas as especificações informais do sistema em contato com o usuário, no caso os engenheiros da usina piloto da CESP. No final, foi elaborado um documento contendo:

- informações sobre o processo físico a supervisionar: gaseificador de madeira, neste caso;
- descrição do sistema atual de medição e supervisão e das suas limitações: no caso, não automatizado, lento, sujeito a erros de leitura, armazenamento em planilhas de difícil uso, etc.;
- especificações informais do sistema a ser desenvolvido: no caso um sistema microcomputadorizado de aquisição de dados com armazenamento em disco flexível, com capacidade de supervisão e monitoração em tela de diversas grandezas (temperaturas, potência elétrica, vazão de gás, etc.) e com alguns parâmetros pré-fixados (tempo de medição, precisão, etc.).

Nesta fase, foram também levantadas, as restrições de custo de investimento do usuário, as expectativas e o grau de experiência dos futuros utilizadores do sistema e os impactos da sua instalação.

Ainda nesta fase, a partir das especificações funcionais informais definidas pelo usuário e da análise das restrições impostas tanto do ponto de vista econômico quanto tecnológico e de recursos humanos, é feito o estudo da realizabilidade do sistema a partir do levantamento das alternativas de solução para o sistema. Uma destas será escolhida e proposta após este exame.

No caso do sistema proposto as restrições são relacionadas:

- ao econômico: custo limitado;
- à disponibilidade de recursos: 2 engenheiros e 2 técnicos de nível médio; sistema microcomputador para desenvolvimento sem as possibilidades gráficas previstas para o projeto;
- ao tecnológico: sistema operacional tipo CP/M; linguagem de fácil aprendizagem pelo u-

suário;

- aos requisitos operacionais: previsão de uma recuperação rápida do sistema no caso de falha na rede de alimentação; previsão de uma futura interligação com outros microcomputadores; interface homem/máquina simples; facilidades para a manutenção do hardware (sinalização), etc.

As especificações funcionais do sistema proposto como solução são contidas num documento descrevendo cada função numa linguagem natural e contendo também a alocação de cada uma das funções ao hardware ou ao software. As principais funções são enunciadas a seguir:

condicionamento dos sinais; filtragem; isolação; amplificação/redução; conversão analógica-digital; entrada de dados não automatizados pelo teclado; cálculo dos valores médios das medidas; armazenamento em disco; supervisão: alarmes na ultrapassagem dos limites de temperatura e de potência elétrica; visualização dos perfis de temperatura nos vários eixos do corpo do gaseificador e das outras medidas indicativas; interface com o usuário com possibilidade de configuração; relatório na impressora; calendário, etc.

PLANEJAMENTO DO HARDWARE E DO SOFTWARE

Ao visto das funções anteriormente alocadas ao hardware e ao software, faz-se uma estimação dos custos, dos recursos humanos envolvidos em cada uma das fases do desenvolvimento, dos recursos materiais e de programação, e do planejamento no tempo do desenvolvimento do sistema (hardware e software).

Para o sistema de aquisição de dados e supervisão, o planejamento no tempo do desenvolvimento do hardware e do software previu uma duração de 22 semanas para cada um; entretanto, o início do desenvolvimento do software ficou posterior ao do hardware por 12 semanas.

ANÁLISE DE REQUISITOS

A partir dos documentos anteriores, são analisados os requisitos do sistema tanto do ponto de vista do hardware como do software e é elaborado um documento contendo as especificações formais. Para tal, utilizam-se as técnicas de análise estruturada (Ross-77), (De Marco-79). O uso exclusivo de uma linguagem natural, nesta fase, é desaconselhado por gerar documentos que podem conter um detalhamento excessivo, ambiguidades, contradições, referencias antecipadas e características não expressas (Meyer-85). A representação gráfica sob forma de um diagrama de fluxo de informação, além de ser de fácil uso, permite evitar estes problemas. Neste diagrama, as bolhas representam os processos de transformação (ou as funções a serem realizadas), as setas indicam as informações de entrada e saída de cada função; os dispositivos externos são representados por caixas e o armazenamento de informações por dois traços paralelos.

Para o caso do sistema de aquisição de dados e supervisão citado, o diagrama de fluxo de informação de uma parte do hardware é representado na figura 3. O refinamento deste dia

grama de fluxo se faz por explosão das bolhas num novo diagrama de fluxo referenciado abaixo da bolha explodida; desta forma, as especificações estão sendo apresentadas utilizando uma hierarquização que facilitará posteriormente o projeto.

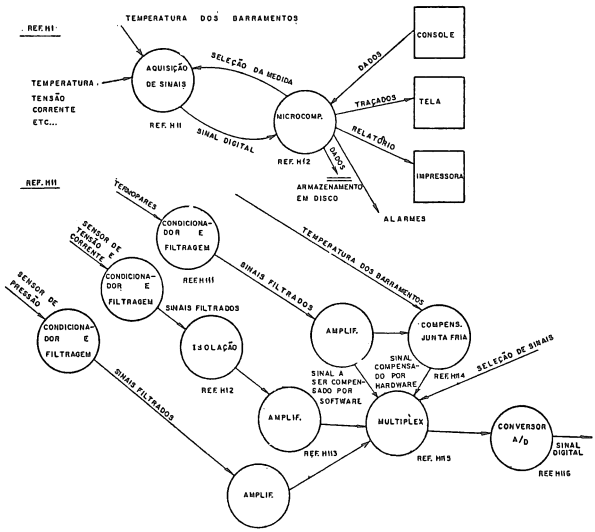


Figura 3: Diagrama de fluxo de informação para parte do hardware do sistema

Para o software, a técnica de análise de requisitos é a mesma; um exemplo desta está apresentado na figura 4.

Define-se também no caso do software, uma estrutura de dados e um dicionário de dados.No nosso exemplo, as grandezas a serem processadas, foram organizadas na forma de uma estrutura hierárquica com os campos temperatura, grandezas elétricas, pressão e características do gás que por sua vez foram também subdivididas; cada campo tem a sua forma própria de organização, por exemplo, no caso do sub-campo temperaturas do corpo do gaseificador segundo um espaço tridimensional (eixo, altura, tempo). Tal estrutura facilitou a proteção e o acesso aos dados a partir das partes do software que precisam deles.

O documento de especificações funcionais formais contém, além dos diagramas de fluxo e da estrutura de dados, uma descrição das várias funções contendo os seguintes ítens: objetivo; interfaces (entradas, saídas, funções em comunicação) e restrições. São definidas também as funções concorrentes e a ordem de prioridade destas.

No final desta fase, é realizada uma etapa de validação onde os vários estados do sistema e todos os caminhos de fluxo de informações são testados. No nosso caso, um diagrama

de estados foi suficiente para tal teste; em casos mais complexos, é necessário utilizar técnicas mais poderosas como a Rede de Petri, por exemplo. A forma estruturada desta fase de análise facilitou a escolha adequada dos módulos e a estruturação a nível de projeto e de implementação, como será apresentado a seguir.

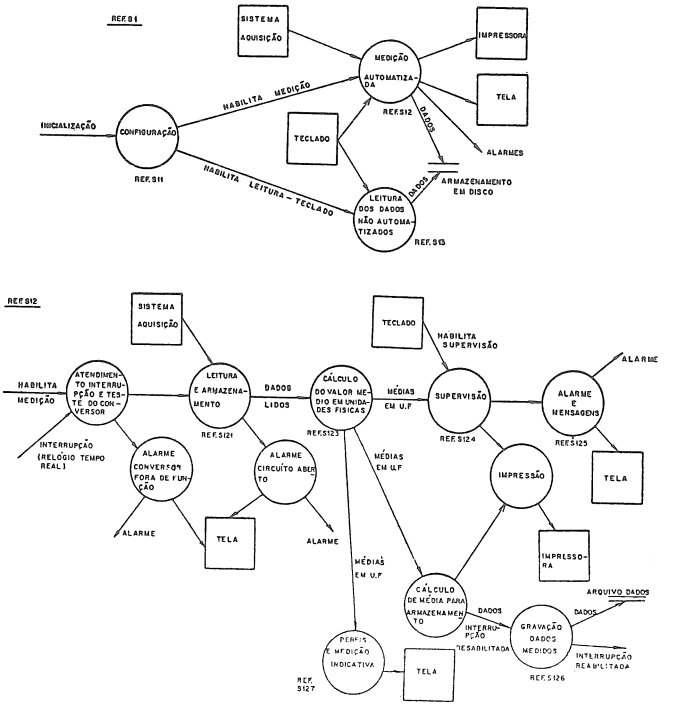


Figura 4: Diagrama de fluxo de informação para parte do software

PROJETO

1. Projeto Preliminar:

A partir dos resultados da fase anterior, os módulos e sub-módulos são definidos utilizando a técnica de projeto estruturado (Yourdon-79).

No projeto preliminar do hardware do sistema de aquisição de dados e supervisão, a partir dos diagramas de fluxo correspondentes e das restrições (como por exemplo, escolha de um barramento em função da futura interligação com outros micros, facilidades para a manutenção da parte material, etc.), procurou-se definir todos os módulos e sub-módulos da parte material. Para cada um destes, descreveu-se a função executada, os interfaces (entradas, saídas, módulos interligados), o procedimento de realização e as restrições ;

estas informações são apresentadas na forma narrativa e na forma de uma representação gráfica onde cada bloco representa um módulo ou um sub-módulo.

No caso do software, o projeto preliminar baseado no diagrama de fluxo de informação visa obter uma estrutura hierárquica com as características de mais alta coesão e de mais baixo acoplamento possível dos módulos e sub-módulos.

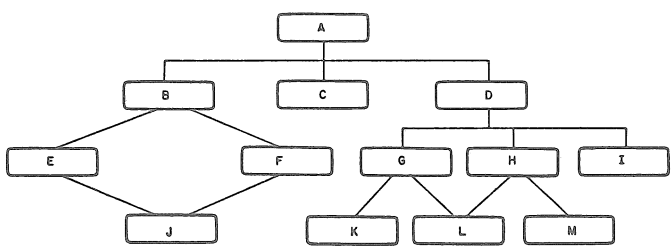
O primeiro passo do projeto consiste em detetar a existência de subsistemas concorrentes (i-e assíncronas) para o sistema descrito pelo diagrama de fluxo de informação. O projeto e a implementação de cada um deles serão feitos de forma independente, tendo sido atribuído também a cada um, um nível e prioridade. Por exemplo, os subsistemas "leitura dos dados não automatizados" e "medição automatizada" (figura 4) são processos concorrentes; atribui-se a maior prioridade ao subsistema "medição automatizada".

A seguir, cada subsistema é decomposto a partir do diagrama de fluxo em módulos e sub-módulos com as características já citadas, procurando também evitar que as informações de um módulo sejam acessíveis a outro, quando não for necessário (Parnas-72), isto é, minimizando o uso de variáveis globais (acessíveis em qualquer lugar do programa).

No final desta decomposição, se obtém uma estrutura hierárquica para cada subsistema. A seguir é realizada a revisão da estrutura do ponto de vista da minimização do acoplamento, do acréscimo da coesão, de possíveis redundâncias e da estruturação dos módulos entre si (fan-in, fan-out).

Cada módulo será também descrito num documento contendo: nome do módulo; objetivo deste; interfaces (entradas, saídas, módulos em comunicação: superordinados e subordinados, forma de chamada); dados globais acessados; estrutura de dados do módulo; algoritmo a ser implementado neste módulo; eventuais restrições.

No caso do sistema de aquisição de dados e supervisão, o subsistema "medição automatizada" tem a estrutura básica apresentada na figura 5 que poderá ser posteriormente refinada.



medição automatizada: A
medição das grandezas físicas: B
cálculo dos valores médios e transformações em unidades físicas: C
supervisão, monitoração e armazenamento: D

leitura do conversor e armazenamento na memória: E
alarme do conversor fora de função: F
atendimento de interrupção e teste do conversor: J
supervisão: G
cálculo de média para relatório e armazenamento: H
traçado dos perfis na tela: I
alarme e mensagem de ultrapassagem na tela: K
impressão: L
gravação de dados: M

Figura 5: Estrutura hierárquica do subsistema "medição automatizada"

2. Projeto Detalhado:

Em seguida, cada módulo de hardware e de software é detalhado por refinamentos sucessivos. No caso do hardware, cada bloco anteriormente definido é refinado introduzindo uma representação simbológica de cada um deles, baseada nas operações eletrônicas básicas tais como integração, comparação, soma, amplificação, etc. No caso do software, o refinamento consiste em reescrever a descrição do módulo feita na etapa anterior até chegar a um pseudo-código que utiliza ao mesmo tempo a sintaxe da programação estruturada com as palavras-chaves (IF-THEN-ELSE; DO WHILE; CASE OF; DECLARE; etc.) e a sintaxe e vocabulário da lingua natural.

IMPLEMENTAÇÃO: MONTAGEM E CODIFICAÇÃO

Nesta fase, são escolhidas as formas práticas de implementação dos módulos de hardware e software anteriormente detalhados; esta fase se caracteriza pela transformação dos blocos de hardware em esquemas contendo o detalhamento de todos os circuitos eletrônicos e na montagem posterior destes; no caso do software, ela corresponde a escrita do código correspondente à pseudo-linguagem da fase anterior.

A escolha da linguagem utilizada é função tanto das possibilidades oferecidas em termos de desenvolvimento (sistema operacional, compiladores, etc.) como das suas características (clareza, estruturação, compactação do código, tipo de entrada-saída, etc.) e das limitações impostas na definição do sistema (memória e tempo de cálculo, código otimizado, possibilidade de fácil modificação pelo usuário e outras exigências). Para o sistema de aquisição de dados e supervisão proposto, alguns módulos foram codificados em Assembly para cumprir com os requisitos de tempo do sistema (módulos ligados a aquisição de dados e ao traçado de gráficos na tela); os outros módulos foram escritos em Basic. O Basic foi escolhido por ser a única linguagem conhecida pelo usuário, permitindo assim uma evolução rápida do programa pela introdução de novos módulos por este. Entretanto as limitações conhecidas desta linguagem em termos de clareza, compactação, tempo de execução levaram a elaborar uma outra versão numa linguagem estruturada, Pascal.

TESTES

1. Testes de Unidade e Integração:

Esta fase deve seguir um plano detalhado elaborado nas fases anteriores (em particular na análise de requisitos e no projeto preliminar); este indicará a sequência de testes, o tipo destes e a forma de integração dos módulos.

No caso do hardware, cada bloco é testado separadamente, levando em conta principalmente critérios de precisão, imunidade ao ruído e faixa passante; em seguida integra-se vários destes blocos para realizar funções maiores.

No caso do software, cada subsistema concorrente será testado independentemente, módulo por módulo, fazendo no final a integração dos mesmos. Para cada módulo, verificar-se-ão as interfaces, a lógica do módulo (percorrendo todos os caminhos condicionais possíveis) e as condições limites destes. O teste de integração realizado a seguir tem dois objetivos:

- testar a estrutura principal do software e a comunicação entre os módulos, por um teste de cima para baixo onde os módulos são representados unicamente pelas suas interfaces;
- integrar os módulos de baixo nível para realizar a função definida por um módulo de nível superior.

Por exemplo, foram testados separadamente os módulos E e F da figura 5 (leitura do conversor e alarme do conversor fora de função) para serem integrados em seguida realizando a função medição (módulo B da figura 5).

No decorrer desta fase, são registrados em documento os resultados dos testes anteriormente planejados; este será de grande utilidade na fase de manutenção do hardware e do software.

2. Teste de Validação:

A validação da parte de hardware e de software é feita de forma independente verificando se os requisitos de cada uma delas são cumpridos, isto é, se todas as funções previstas são realizadas com as características desejadas (como por exemplo, a precisão no caso do hardware).

"ASSEMBLAGEM" E TESTE DO SISTEMA TOTAL

Os frequentes contatos efetuados no decorrer das várias fases do desenvolvimento de hardware e de software facilitaram a "assemblagem". Nesta fase, serão testadas principalmente as partes do sistema que dependem ao mesmo tempo do hardware e do software como por exemplo a parte do sistema que realiza a aquisição de dados e a que permite a leitura a partir do teclado dos dados não automatizados.

Uma vez a "assemblagem" acabada, é realizado o teste final do sistema: em laboratório e

em seguida no próprio local da instalação final deste. Neste teste são verificados o cumprimento de todos os requisitos funcionais do sistema. a compatibilidade deste com a planta. o desempenho deste em operação e a conformidade da documentação que acompanha o sistema. Uma vez esta fase de testes terminada, o sistema é considerado aceito e pode ser utilizado normalmente pelo usuário.

CONCLUSÃO

Neste trabalho foi apresentada uma metodologia para o desenvolvimento de um sistema de aquisição de dados e supervisão. Ela foi testada com sucesso num sistema a ser instalado num gaseificador de madeira de uma usina piloto de produção de metanol.

Esta metodologia contribui para se obter um produto final de melhor qualidade, mais confiável e melhor documentado e em consequência com maiores possibilidades de modificação e evolução. Constatou-se também algumas dificuldades na aplicação desta metodologia que deverão ser levadas em conta em trabalhos futuros. A fase de definição do sistema deverá ser mais aprofundada, apesar das dificuldades de comunicação com o usuário. Os possíveis atrasos no fornecimento de material deverão ser melhor avaliados e levados em conta no planejamento no tempo. Em caso da existência de um grande número de subsistemas concorrentes, é necessário utilizar as técnicas de análise e validação apropriadas tais como por exemplo as Redes de Petri.

Deve-se notar que o uso da estruturação a nível de análise, projeto e implementação melhorou sensivelmente a qualidade do produto e o tempo envolvido no desenvolvimento.

Um fator de melhoria na "assemblagem" do sistema deveu-se ao fato que, por razões de escassez de recursos humanos, os projetistas de hardware tiveram também envolvidos no trabalho de desenvolvimento do software e vice-versa. Entretanto, em caso de desenvolvimento de sistemas maiores, tal possibilidade desaparece e só pode ser substituída por contatos regulares e organizados entre as duas equipes: de hardware e de software.

BIBLIOGRAFIA

- DE MARCO T. - "Structured Analysis and System Specification" - Prentice-Hall, 1979.
- GOMAA H., LUI J., WOO P. - "The Software Engineering of a Microcomputer Application System" - Software: Practice and experience, vol.12, pp. 309-321, 1982.
- MEYER B. - "On Formalism in Specifications" - IEEE Software, vol.2, nº1, pp.6-36, jan. 1985
- NETO J.J. - "Introdução a Engenharia de Software e Metodologia de Programação" - 1º CONAI - Tutorais, pp. 119-146, jul. 1983.
- PARNAS D.L. - "On the Criteria to be used in Decomposing a System into Modules" - Communications ACM, pp. 1053-1058, dez. 1972.
- PRESSMAN R.S. - "Software Engineering: A Practitioner's Approach" - Mac-Graw-Hill-1982
- ROSS D., SCHOMAN K. - "Structured Analysis for Requirements Definition" - IEEE Trans. on Software Engineering, vol.3, nº1, jan.1977, pp.6-15.
- YOURDON E., CONSTANTINE L. - "Structured Design" - Prentice-Hall, 1979.

UMA FERRAMENTA PARA GERAÇÃO DE DIÁLOGOS

M.V. SILVA *, L.P. MAGALHÃES **

SUMÁRIO

O PROCESSADOR DE DIÁLOGO, uma ferramenta para definição, teste de protótipos e geração de estruturas de diálogos otimizadas é definido. Para apoio ao tratamento gráfico, é justificada a utilização de um Núcleo Gráfico Padrão e, em decorrência, a existência de um GERENCIADOR DE DIÁLOGO.

ABSTRACT

A DIALOGUE PROCESSOR, a tool for definition, prototyping and generation of optimized dialogue structures, is defined. The utilization of a Standards Graphics Package for Dialogue Processor support in graphics treatment is justified; as a consequence, the necessity of a DIALOGUE MANAGER is shown.

* Eng. Eletrônico e de Telecom.,PUC/MG, 1981,
atualmente inscrito no programa de mestrado do DEE-FEC-UNICAMP

** Dr. Ing., T.H. Darmstadt, Alemanha, 1981,
atualmente Prof.-Ass.-Dr. DEE-FEC-UNICAMP

UNICAMP-FEC-DEE; CP 6122; CEP 13100 - CAMPINAS - SP; Tel.:(0192) 39-1395

1-INTRODUÇÃO

O presente trabalho apresenta o PROCESSADOR DE DIÁLOGO, já discutido anteriormente (/SILV 84/), fornecendo uma descrição funcional e topológica do mesmo. O Processador de Diálogo é uma ferramenta de apoio à geração de diálogos otimizados. Baseia-se na premissa de separar-se a atividade de diálogo da atividade a ser controlada pelo diálogo. Sendo a comunicação homem-máquina tratada isoladamente da aplicação, é possível utilizar um especialista em cada área para tratar da atividade correspondente.

São discutidas as principais características desta ferramenta de software para auxílio no projeto, análise e implementação ("prototyping") de diálogos homem-computador. Tal ferramenta deve ser independente da aplicação específica, da linguagem de programação utilizada pela aplicação e dos dispositivos a serem acionados para entrada e saída de dados. São características suas, a possibilidade de definir e alterar a definição de um diálogo, supervisionar a execução, coletar dados sobre o desempenho, analisar estes dados e promover uma adaptação do diálogo ao operador específico. Para a representação do diálogo, o Processador tem como recurso a Linguagem de Especificação de Diálogos (LED) que permite a definição da estrutura ou das estruturas de qualquer diálogo. Esta linguagem é baseada no método de representação de estruturas de diálogo através de Células de Diálogo (/SILV 85/).

O item 2 deste trabalho apresenta o Processador de Diálogo. O item 3 discute a utilização de um Núcleo Gráfico Padrão para suporte gráfico do Processador de Diálogo. O item 4 apresenta as conclusões.

2-O PROCESSADOR DE DIÁLOGO

A separação entre a aplicação e o diálogo que a controla nos sistemas interativos de computação, com a comunicação usuário-sistema sendo realizada por uma camada independente do programa de aplicação e do pacote gráfico de suporte, traz diversos benefícios, entre os quais (/GIIT 83/) :

- fomenta um trabalho interdisciplinar, com especialistas de aplicação, de núcleo gráfico, de interação, cada um realizando uma parte do projeto do software;

- permite ao programador de aplicação projetar suas rotinas considerando que rotinas de interação previamente implementadas estão ao seu dispor para realizar a troca de informações com o usuário;

- permite exploração de várias características de dispositivos físicos e de técnicas de interação para atender aos fatores humanos do operador;

- traz economia de custos, facilitando a construção da interface de

usuário, permitindo o seu aprimoramento para casos específicos e a sua reutilização por outras aplicações;

- permite consistência através de várias aplicações que podem fazer uso da mesma interface.

Entretanto, nenhum sistema de diálogo pode ser implementado, a priori, suficientemente otimizado para o tipo de aplicação em questão. Assim, surge uma necessidade de avaliação de desempenho da interação operador-sistema e de uma adaptação da estrutura do diálogo às características específicas de um determinado operador e de uma determinada aplicação.

Para suprir esta necessidade, deve-se obter recursos para a definição de estruturas alternativas para um mesmo diálogo, para a alteração de passos de uma determinada estrutura, para controle da execução do diálogo, para coleta de dados sobre o desempenho, para a avaliação destes dados e finalmente para a substituição de uma estrutura de diálogo por outra. Repetindo-se o processo de supervisão de execução e de coleta e avaliação de dados, pode-se determinar qual a estrutura mais eficiente, obtendo-se então uma otimização do trabalho a ser executado pelo operador quando utilizando o sistema.

Todas estas tarefas são referidas coletivamente como "prototyping", entendido como sendo a simulação da execução de um diálogo, podendo acarretar em monitoramento, análise e adaptação do mesmo a uma situação específica e na posterior geração de um produto final, constando da aplicação e seu diálogo otimizado.

Assim, é aqui proposta uma ferramenta, o PROCESSADOR DE DIÁLOGO, que pretende auxiliar o projetista na produção de diálogos, permitindo definição, "prototyping" e geração de um diálogo otimizado. A ferramenta possibilita ao projetista executar um projeto estruturado e, desde que este projetista tenha em mente as considerações de fatores humanos, ele está capacitado a traduzir em seu diálogo um sistema fácil e agradável ao usuário. Uma vez identificada a estrutura ótima para o diálogo entre um operador e uma aplicação específicos, o Processador de Diálogo está apto a gerar seu produto final, qual seja um sistema de aplicação envolvido com seu diálogo otimizado, onde todos aqueles componentes do Processador, utilizados no "prototyping" mas não mais necessários à execução do sistema interativo são subtraídos.

Pode-se visualizar três fases na utilização do Processador de Diálogo : definição, "prototyping" e geração, apresentadas a seguir.

2.1-Definição de Diálogos

Na fase de Definição, um diálogo específico é descrito através dos recursos oferecidos por uma Linguagem de Especificação de Diálogos (LED) (/SILV 85/), baseada no método de representação de estruturas de

diálogo por Células de Diálogo (/SILV 84/). Assim, cada passo em cada interação fica a cargo de uma Célula de Diálogo, que na prática, corresponde a uma rotina. A estrutura do diálogo estará então descrita através destas rotinas. Sendo a Linguagem de Especificação de Diálogos, uma extensão do FORTRAN-77, não são providos compiladores que codifiquem as descrições em forma objeto.

Assim, deve-se utilizar um pré-processador que leia as Células de Diálogo, interprete-as e gere rotinas em FORTRAN-77, através do mapeamento das primitivas extendidas da LED às primitivas básicas do FORTRAN-77. Após este pré-processamento, as rotinas podem ser submetidas ao compilador FORTRAN-77 que se encarrega de gerar rotinas em linguagem objeto, que são ligadas para obtenção do programa executável. Faz parte da função do pré-processador a análise do código que permita a detecção de erros de digitação quando da criação dos arquivos e de falhas na observação das regras da linguagem.

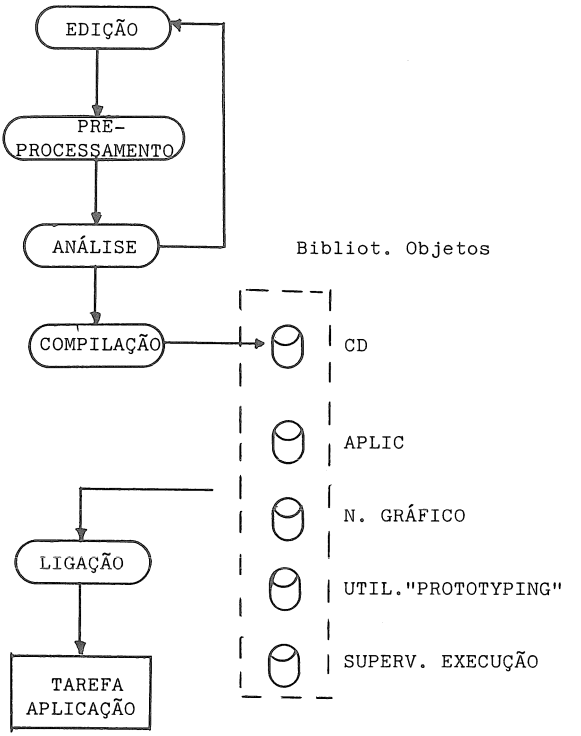


fig. 1 : Fase de definição.

Cada conjunto de células de diálogo capaz de realizar uma interação com o operador é reunido sob a forma de um subprograma (subprograma diálogo). Assim, pode-se ter diversos subprogramas diálogo capazes de atender a cada pedido de dado realizado pela aplicação. A otimização do diálogo está em se determinar, para cada interação, qual é o subprograma mais adequado.

Além das Células de Diálogo, são necessários à execução do diálogo, o programa de aplicação que fará uso do diálogo (ou um simulador dele, caso se pretenda testar o diálogo separadamente ou previamente em relação ao desenvolvimento do programa de aplicação), as rotinas de um núcleo gráfico, que dão apoio à geração e manipulação de figuras para a comunicação com o operador, o Supervisor de Execução que controla a execução do "prototyping" e rotinas (na forma de FUNCTION's) de apoio ao "prototyping" (para coleta de dados, análise, etc, algumas providas pelo implementador do Processador de Diálogo e outras pelo próprio autor de diÍogos.

A tarefa aplicação, definida como sendo o programa de aplicação ligado ao supervisor de execução, aos subprogramas de diálogo, às rotinas do núcleo gráfico e aos utilitários de "prototyping", atrvés do uso do ligador do sistema operacional do computador hospedeiro.

A figura 1 apresenta a sequência de atividades desde a criação das células de diálogo até a obtenção da tarefa aplicação.

2.2-"Prototyping"

Uma vez obtida a tarefa aplicação, deve ser realizado o "prototyping". A tarefa aplicação é executada e o supervisor de execução é acionado toda vez que a aplicação desejar se comunicar com o usuário. O supervisor se configura então como a ligação entre a aplicação e o diálogo, incumbindo-se de acionar um dos subprogramas diálogo providos para aquela interação e passar-lhe o controle de execução. O subprograma é executado e, nos locais definidos pelo autor de diálogos, são ativadas funções de "prototyping" (providas na forma de FUNCTION's) que realizarão coleta e análise de dados (e outras atividades previstas). Quando o controle é devolvido ao supervisor de execução é acionado outro subprograma diálogo para a mesma interação e repete-se o processo que permite então a avaliação dos subprogramas diálogo projetados para aquela interação. A seguir, o controle é devolvido ao programa de aplicação que retoma sua execução. A cada nova interação, ou seja, a cada nova necessidade de comunicação entre o programa de aplicação e o operador, um novo conjunto de subprogramas diálogo é executado, um subprograma por vez, permitindo avaliar os diversos subprogramas providos para cada interação. Ao final da execução da tarefa aplicação, pode-se determinar, pelos dados coletados, qual subprograma é apropriado para cada interação com a aplicação em questão e o conjunto destes subprogramas constitui o diálogo otimizado para aquela aplicação. A figura 2 apresenta o diagrama

da tarefa aplicação sendo executada, onde são exemplificadas m interações com n possibilidades de sequências de diálogo diferentes para cada uma das interações.

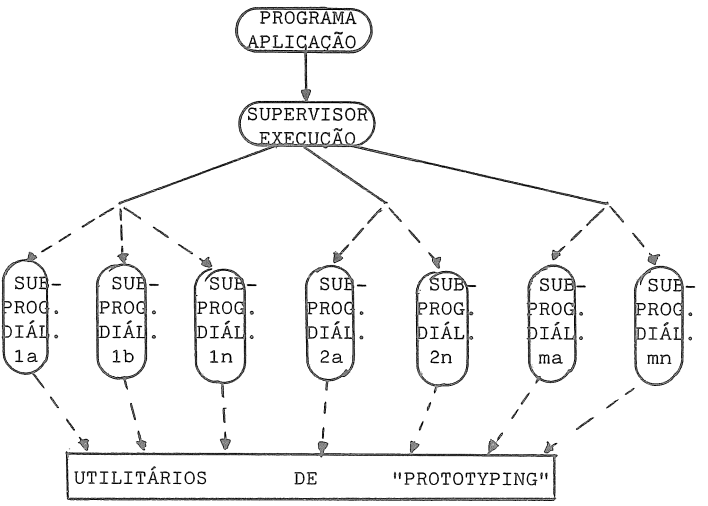


fig. 2 : Fase de "prototyping"

2.3-Geração do Diálogo Otimizado

A geração é realizada após o elemento LIMPADOR processar os subprogramas diálogo selecionados na fase anterior e eliminar das células de diálogo correspondentes todas as primitivas específicas de "prototyping", desnecessárias à operação normal do diálogo.

As células de diálogo resultantes são novamente pré-processadas, compiladas e ligadas, agora ao programa de aplicação, ao supervisor de execução e às rotinas do núcleo gráfico. Deve-se notar que o supervisor de execução agora utilizado é uma versão simplificada daquele utilizado para o "prototyping", já que não são mais necessários os testes e controles para acionar diversas sequências de diálogo para atender a cada solicitação de interação efetuada pela aplicação. Já foi identificada a sequência ótima para cada interação e o supervisor simplesmente faz a interface entre a aplicação e as sequências ótimas. A operação de obtenção da tarefa aplicação ótima é ilustrada na figura 3.

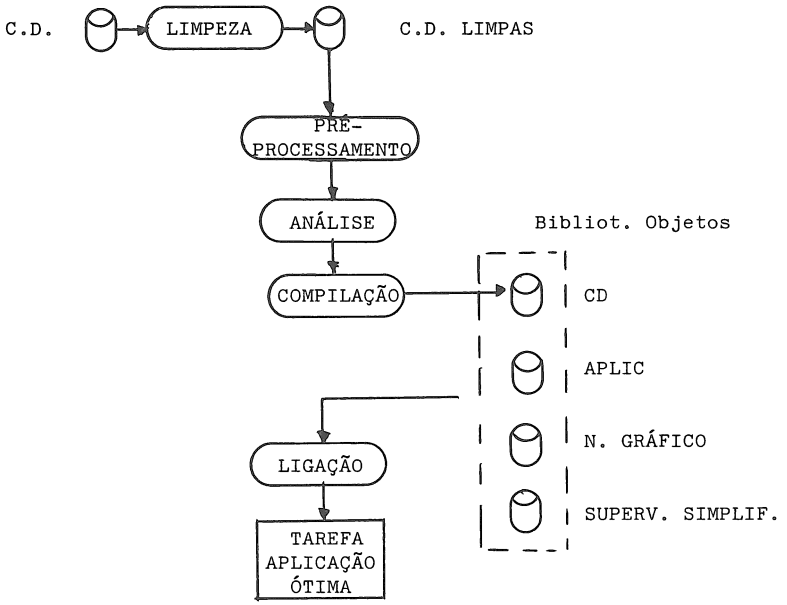


fig. 3 : Fase de geração.

3-USO DE UM NÚCLEO GRÁFICO PADRÃO NO APOIO AO PROCESSADOR DE DIÁLOGO

O Supervisor de Execução (figura 2) será enfocado mais detalhadamente. Para tal, ele é analisado como sendo constituído por três elementos : um Supervisor de "Prototyping", um Selecionador de Sequências de Diálogo e um Gerenciador de Recursos Gráficos.

O Selecionador de Sequências é responsável por, mediante uma solicitação de interação da aplicação com o operador, determinar quais são (no "prototyping") ou qual é (na execução normal da tarefa aplicação ótima) o(s) subprograma(s) diálogo previsto(s) para aquela interação específica.

Em termos de implementação, o diálogo apresenta-se (conforme pode ser visto na figura 2) como uma árvore, onde o Selecionador de Sequências é a raiz. Cada ramo imediatamente abaixo do Selecionador corresponde a uma necessidade de comunicação do programa de aplicação com o operador. Mas, podem ser oferecidas diversas possibilidades de realização de cada uma destas comunicações. Assim, a cada ramo deixando a raiz da árvore pode corresponder uma subárvore, cujos ramos representam

as possibilidades de atendimento daquela necessidade de comunicação. A decisão por um ramo ou outro depende de valores de parâmetros passados pelo programa de aplicação ao Selecionador que lhe permitem determinar qual o tipo de necessidade de comunicação e então qual ramo deve ser escolhido.

O Supervisor de "Prototyping" tem sua atividade restrita à fase de "prototyping", já que, em operação normal do diálogo, ele não é necessário. Sua função é a de selecionar ordenadamente, para cada solicitação de interação da aplicação em relação ao operador, cada sequência de diálogo, na forma de subprograma diálogo, provida para atender àquela interação, permitindo a avaliação de todas elas.

Antevendo a implementação, o Supervisor de "Prototyping" agiria como uma estrutura do tipo CASE, segundo a qual, condições seriam manipuladas e testadas implicando no acionamento de uma rotina (subprograma diálogo) para realizar a interação com o operador. Após o retorno do controle de execução ao Supervisor de "Prototyping", as condições são alteradas permitindo o acionamento de outra rotina, e o processo se repete até que todas as rotinas providas para a execução do diálogo correspondente a um dado pedido de interação do programa de aplicação com o operador sejam executadas.

Associando à descrição do Selecionador, o Supervisor de "Prototyping" estaria logo abaixo na árvore e se incumbiria de executar cada ramo da subárvore correspondente ao ramo inicial escolhido pelo Selecionador.

O Gerenciador de Recursos Gráficos surge em consequência :

- da utilização de recursos de núcleos gráficos padrões pelo diálogo; tem então o objetivo de adequar os recursos do núcleo gráfico às necessidades do diálogo;

- da necessidade de supervisão dos elementos de comunicação definidos nas células de diálogo;

- do desejo de proporcionar à aplicação acesso aos recursos do núcleo gráfico.

A primeira consequência vem da necessidade de execução das seguintes funções :

- gerenciamento das transformações de coordenadas das informações gráficas definidas pelo diálogo e mapeadas ao sistema gráfico;

- gerenciamento de segmentos; mapeamento dos elementos de comunicação aos segmentos gráficos do sistema gráfico, com o necessário provimento da identificação de cada segmento a ser criado;

- gerenciamento do efeito das funções de controle do sistema gráfico para adequá-las às necessidades do diálogo;

- mapeamento dos parâmetros necessários à ativação de primitivas do núcleo gráfico a partir das primitivas gráficas da LED.

A segunda consequência atende às seguintes funções :

- atribuição de valores para a identificação dos elementos de comunicação criados;

- controle do tempo de vida de cada elemento. Finda a duração definida, o elemento deve ser explicitamente deletado;

- vinculação das figuras aos elementos correspondentes, conforme definido na célula. Se um elemento é alterado, a figura correspondente também deve ser alterada.

- manutenção de informações sobre o desenrolar do diálogo, tais como:

  - estado corrente do diálogo : inativo, ativo (existe célula em execução);

  - identificação da célula em execução;

  - lista das células correntemente ativas (aquela em execução é a mais interna na hierarquia);

  - estado de erro (se e onde ocorreu);

  - tabela de correlação entre a identificação de um dispositivo gráfico pelo núcleo gráfico e a identificação daquele dispositivo pelo gerenciador de diálogo;

  - lista dos elementos definidos, contendo o identificador, o tipo de elemento, os atributos correntes e a células em que cada elemento está definido.

A terceira consequência tem a ver com o oferecimento de facilidades gráficas à aplicação. Pode ser desejado permitir à aplicação acesso às facilidades gráficas do núcleo que dá suporte ao Processador de Diálogo. Ocorre, entretanto, que restrições devem ser feitas a este acesso, para que não haja conflitos de acesso entre a aplicação e a camada de diálogo, o que prejudicaria a representação final. A solução apresenta-se como sendo a utilização uniforme do núcleo gráfico tanto por parte do diálogo quanto da aplicação, através da definição de uma primitiva do Gerenciador para cada recurso do núcleo gráfico, a ser mapeada a estes recursos após tratamento das restrições pelo Gerenciador.

As funções deste Gerenciador são discutidos em maior grau de

detalhe em /SILV 85/.

4-CONCLUSÃO

As vantagens de uso do Processador de Diálogo podem ser resumidas como :

- formalizar, ou padronizar, a arte de projetar diálogos. Um tratamento padronizado possibilita o rápido aprendizado da técnica;

- oferecer uma Linguagem de Especificação de Diálogos clara, legível e objetiva, mapeada sobre uma linguagem padrão;

- permitir testar diversos diálogos, compará-los e selecionar o de melhor efetividade, adaptando então o diálogo a um operador e a uma aplicação específicos;

- permitir projeto modular, onde cada camada do sistema interativo fica a cargo do especialista correspondente;

- possibilitar a utilização de informações gráficas no diálogo entre operador e aplicação, o que traz aumento de efetividade;

- permitir economia de custos, pois o Processador pode ser utilizado por diversas aplicações;

- permitir explorar características de dispositivos e técnicas de interação com o objetivo de tornar a interface "amigável" ao usuário.

A decisão de utilização do nucleo gráfico padrão como apoio ao Processador de Diálogo veio do desejo de se ter uma interface com o mundo exterior portátil e consistente. Esta decisão trouxe a necessidade de se definir um elemento Gerenciador de Recursos Gráficos, que se responsabilize por supervisionar a utilização dos recursos do núcleo gráfico adequando-os às necessidades da camada de diálogo, e para permitir às aplicações que se utilizam do Processador de Diálogo, o acesso às mesmas facilidades gráficas oferecidas ao diálogo.

5-REFERÊNCIAS BIBLIOGRÁFICAS

/GIIT 83/ - GRAPHICAL INPUT INTERACTION TECHNIQUE WORKSHOP SUMMARY - Computer Graphics - 17 (1) - JAN 1983

/SILV 84/ - Silva, M.V.; Magalhães, L.P.; "Processadores Universais de Diálogo : Formas de Representação do Diálogo" - Anais do IV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO - JUL 1984

/SILV 85/ - Silva, M.V.; "PROCESSADOR DE DIÁLOGO : Uma Ferramenta para Geração de Sistemas Interativos" - Tese de Mestrado - UNICAMP - Campinas - SP - ABRIL 1985

METODOLOGIA DE CONSTRUCCION DE PROGRAMAS A PARTIR DE ESPECIFICACIONES FORMALES

M. RUKOZ *

RESUMEN

El objetivo de este trabajo es la elaboración de una metodología de construcción de programas que permita de una forma sistemática elaborar programas confiables.

La metodología se basa por una parte en la utilización de operadores genéricos, que permiten hacer abstracciones operacionales y por otra parte en la representación algoritmica de Tipos de Datos Abstractos, que permitan hacer abstracciones en la estructura de datos.

ABSTRACT

This paper intented to develop a program construction methodology in order to elaborate reliable software in a systematic manner. Methodology is based on utilization of generic operator and algorithmical representation of abstract data types, that make possible the action and object abstractions respectively.

* Licenciada en Computación (Vla, 1981)
Universidad Central de Venezuela, U.C.V.,
Facultad de Ciencias, Escuela de Computación,
Apartado 47002, Caracas 1041-A, Venezuela.
Tlf. (02) 6627233

INTRODUCCION

Dentro del proceso de diseño e implantación de sistemas de computación, uno de los principales problemas es la reducción de complejidad que la persona debe afrontar en un momento dado para la solución del mismo. Una vía para resolver esto es a través de la abstracción, esto es, desarrollar modelos que revelen las propiedades relevantes e ignoren los detalles irrelevantes del sistema.

La resolución algoritmica de problemas implica el diseño de un algoritmo, basado en cierta metodología que permita una solución (si existe) y en la decisión de la organización de la información relevante al mismo. Estos dos factores están íntimamente relacionados, de manera tal que las decisiones acerca de la estructuración de los datos requiere conocer del algoritmo, así como el algoritmo depende de tal estructuración.

Así podemos distinguir dos puntos en la solución algoritmica de un problema donde se puede hacer uso de la abstracción: el algoritmo y la estructura de datos. Surgen así dos mecanismos de abstracción: abstracción operacional y abstracción de datos, según (Bert 82a) 'The action abstraction' y "The object abstraction' respectivamente.

El objetivo de este trabajo es la elaboración de una metodología de construcción de programas que permita de una forma sistemática elaborar programas confiables.

La metodología se basa por una parte en la utilización de operadores genéricos, que permiten hacer abstracciones operacionales y por otra la representación algoritmica de tipos de Datos Abstractos, que permiten hacer abstracción en la estructura de datos.

En dicha metodología se especifica un problema funcionalmente, lo cual permite un análisis completo del mismo a nivel de especificaciones, para luego transformarlo en un programa algoritmico utilizando esquemas de programación establecidos.

En la primera sección, se plantea un lenguaje que permita especificar operadores genéricos y Tipos de Datos Abstractos, y como se puede especificar funcionalmente un problema utilizando dicho lenguaje. En la segunda sección se dan esquemas de programas algoritmicos para cada uno de los constructores del lenguaje dado y por último en la tercera sección se define la metodología.

1. PRESENTACION DEL LENGUAJE DE ESPECIFICACION

Con el fin de obtener especificaciones matemáticamente correctas y relativamente fáciles de elaborar, utilizaremos el lenguaje de especificación y programación LPG (Lenguaje de Programación Genérica) este es presentado en (Bert 83) y actualmente está implementado en el Laboratorio IMAG de la Universidad de Grenoble, Francia.

Este lenguaje está basado en las especificaciones algebráicas de Tipos de Datos Abstractos (Gut 78a), (Gut 78b) y permite definir de una manera abstracta los tipos de datos y el conjunto de operadores.

En LPG tanto los tipos de datos como los operadores pueden ser genéricos, es decir, parametrizados por otros tipos llamados tipos formales. Un tipo formal está caracterizado por

un conjunto mínimo de operadores que deberá poseer el tipo efectivo. Este conjunto mínimo de operadores puede ser identificado con el nombre de propiedad y se definen semánticamente a través de axiomas ecuacionales.

Además de los operadores genéricos LPG presenta otros constructores básicos que permiten especificar problemas, estas son: la condición, la composición y la sustitución.

A continuación se muestran como se especifican algebráicamente los tipos de datos abstractos, la definición de los constructores básicos y como se definen los operadores genéricos en LPG.

1.1 Definición de Tipo de Datos Abstractos en LPG

Tipo nombre-del-tipo

opns
  { identificación sintáctica de los operadores}

Vars
  { declaración de variables }

equs
  { identificación semántica de los operaciones}

1.2 Constructores básicos de LPG

Condición: Si b ENTONCES A SINO B FSI

Composición: F; G

Sustitución: SEA w = C EN B(w) fSEA

donde: b, A, B, F, C son constructores de LPG; w es una variable; B(w) es un constructor que contiene w.

1.3 Definición de Operadores Genéricos en LPG

Enriq : nomb-del-enriq exige nombre-de-propiedad ($t_1,..t_n$ opns $0_1..0_n$)

opns
  {identificación sintáctica de los operadores }

vars
  {declaración de variables }

equs
  {lista de ecuaciones que identifican semánticamente a los operadores }

fin

donde: $t_1,..t_n$ son los tipos efectivos sobre los cuales se exige la propiedad

$0_1,..0_n$ son los operadores efectivos

Prop nombre-de-propiedad sobre $t_1, \ldots, t_n$ (tipos formales)

opns

{identificación sintáctica de los operadores formales}

vars

{declaración de variables}

equs

{lista de ecuaciones que identifican semánticamente a los operadores}

fin

Para expresar funcionalmente un problema se le han asociado a los constructores básicos y a los operadores genéricos de LPG una forma funcional. En la tabla I se muestran las formas funcionales de los diferentes operadores a considerar.

Las formas funcionales permiten hacer manipulaciones en las especificaciones, en la tabla II se presentan diferentes reglas que permiten pasar de una forma funcional a otra, en estas reglas se ha hecho abstracción de las variables que utiliza una determinada forma funcional.

FORMAS FUNCIONALES DE LOS CONSTRUCTORES DE LPG

Condición Si b ENTONCES A sino B FSI (x) = =

Si b(x) ENTONCES A(x) sino B(x) FSI

Composición F o G (x) = = F(G(x))

Sustitución SEA w = A EN B(w) FSEA (x) = = B(A(x), x)

Operadores Genéricos

- phi $\{f_1, f_2\}$ $(x,y) = = (f_1(x), f_2(y))$
- mu $\{f_1, f_2\}$ $(x,y) = = (f_1(f_2(x), y))$
- psi $\{f_1, f_2\}$ $(x,y) = = (f_1(x), f_2(x))$
- iter $\{f_1, f_2\}$ $(x, \langle y_1, y_2, \ldots y_n \rangle) = =$ SEA $w = f_2(x, y_1)$
  EN $\pi_1(w) \rightarrow$ iter $\{f_1, f_2\}$ $(\pi_2(w), \langle y_2, \ldots, y_n \rangle)$
  FSEA

  iter $\{f_1, f_2\}$ $(x, \langle : t_1 \rangle) = = f_1(x)$
- arra $\{f_1, f_2, f_3\}$ $(x_1, \langle y_1, y_2, \ldots y_n \rangle)$ SEA $w = f_2(x_1, y_1)$
  EN $f_3(w,$ arra $\{f_1, f_2, f_3\}$ $(\pi_2(w), \langle y_2, \ldots y_n \rangle)$
  FSEA

  arra $\{f_1, f_2, f_3\}$ $(x_1, \langle : t_1 \rangle) = = f_1(x)$

TABLA I (Continúa)

TABLA I (Continuación)

- alpha $\{f\}(\langle x_1,\cdots,x_n\rangle) = = \langle f(x_1),f(x_2),\cdots,f(x_n)\rangle$
- $\partial'\{f,x\} = = f(x)$
- $\$'\{f,x\}\ (y) = = f(x,y)$
- $C'\{x\}\ (y) = = x$
- $S\{i\}(\langle x_1,x_2,\cdots x_n\rangle) = = x_i$ si $1 \leq i \leq n$, sino indefinido
- $hom\{c,f\}(\langle x_1,x_2,\cdots x_n\rangle) = = f(x_1,hom'\{c,f\}(\langle x_2,\cdots,x_n\rangle))$
  $hom\{c,f\}(\langle :t\rangle) = = c$

NOTA: $\langle :t\rangle$ es la secuencia vacía de elementos de tipo t
! 1 y ! 2 son funciones definidas de la manera siguiente:

$!1: (t_1,t_2) \to t_2$; $!2: (t_1,t_2) \to t_2$; a: t_1, b: t_2

!1(a,b) = = a y !2(a,b) = = b

REGLAS DE TRANSFORMACION

1. id o g = = g o id = = g
2. (f o g) o h = = fo(g o h)
3. Si b ENTONCES y SINO q FSI oh = = Si b o h ENTONCES y o h SINO q o h FSI
4. h o Si b ENTONCES r SINO q FSI = = Si b ENTONCES h o r SINO h o q FSI
5. Si b ENTONCES Si b ENTONCES p SINO q FSI SINO r FSI = =
   Si b ENTONCES p SINO r FSI
6. Si b ENTONCES r SINO q FSI = = Si NO(b) ENTONCES q SINO r FSI
7. Si C'{VERDAD} ENTONCES r SINO q FSI = = r
8. Si C'{FALSO} ENTONCES r SINO q FSI = = q
9. Si b ENTONCES C {VERDAD} SINO p FSI = = OR o psi {b,p}
10. ! 1 o Psi $\{f_1,f_2\} = = f_1$
11. C'{h} o g = = C'{h}
12. ! 2 o psi'$\{f_1,f_2\} = = f_2$
13. ! 1 o phi'$\{f_1,f_2\} = = f_1$ o !1
14. !2 o phi'$\{f_1,f_2\} = = f_2$ o !2
15. phi'$\{f_1,f_2\}$ = = psi'$\{f_1$ o !1, f_2 o !2$\}$
16. phi'$\{f_1,f_2\}$ o psi'$\{f_3,f_4\}$ = = psi'$\{f_1$ o f_3, f_2 o $f_4\}$
17. phi'$\{f_1,f_2\}$ o phi'$\{f_3,f_4\} = =$ phi'$\{f_1$ o f_3, f_2 o $f_4\}$
18. mu'$\{f_1,f_2\}$ o psi'$\{f_3,f_4\} = = f_1$ o psi'$\{f_2$ o f_3, $f_4\}$

TABLA II

A cada constructor funcional de LPG (constructores básicos y operadores genéricos) se le ha definido un esquema algoritmico que permita el paso casi mecânico de las especifica - ciones al algoritmo final del problema.

A continuación se presentan los diferentes esquemas de programación:

```
Esq   (Si  b  ENTONCES  A  SINO B  FSI)
ent   x: tl  Sal  R: t2
inst  Si Sal (esq (b(x))) entonces  R: = sal(esq(A(x)))
                          Sino  R: = sal (esq(B(x)))  fsi
Esq (f(g(x))
ent  x:tl  sal R:t2  Var  y:t
inst  y:= sal(esq(g(x))); R:= sal(esq(f(y)))
Esq  (SEA  w = A EN  B(w)  FSEA )
ent  x:t  sal  R:t2
inst  w:= sal (esq(A(x)) ; R:= sal(esq(B(w,x))
Esq (phi { f1,f2 } )
ent   X:tl, y:t2   sal  R1:t3 , R2:t4
inst  R1: = sal (esq(f1(x))) ; R2:= sal (esq(f2(y)))
Esq (mu { f1,f2 } )
ent  X:tl  y:t2  sal  R:t3  var  X2:t3
inst  X2:= sal (esq(f2(x))); R:= sal(esq(f1(X2,y)))
Esq  (psi { f1,f2 } )
ent  X:tl  sal  R1:t2, R2:t3
inst  R1: = sal(esq(f1(X))) ; R2 : = sal(esq(f2(X)))
Esq  (iter { f1,f2 } )
ent  P1:t3 ; S1: seq < tl > sal  S2 : seq < t2 >
var  X:tl, SF2: seq < t2 >,PF2 : t3
inst  PF2:= P1;
     mientras  no (nul(S1)) hacer
        X:=PRIMERO(S1) ; S1:=RESTO (S1)
        (SF2,PF2) : = sal (esq(f2(PF2,X))) ;
         S2 : = S2 + SF2
     Fmientras
     S2: = S2 + sal (esq(f1(PF2)))
```

```
Esq (arra { f1,f2,f3} )
  ent  P1 : t3   S1 : seq < t1 > ; sal   S2 :  seq < t2 >   P2:t3
  var  SF2: seq < t2 > ,  PF2 : t3   X : t1
  inst   S2 : = nul ; P2 : = P1 ; PF2 : = P1 ;
         mientras   no(nul(S1))  hacer
            X : =  PRIMERO (S1) ; S1 : = RESTO (S1)
            (SF2,PF2) : = sal (esq(f2(PF2,X))
            (S2,P2) : =   sal (esq(f3(S2,P2,SF2,PF2)))
         fmientras
         (S2, P2 ) : = sal (esq(f1 (P2)))

Esq (Alpha { f } )
ent  S1 : seq < t1 > sal  S2: seq < t2  >, var   X:t1
inst  S2: = nul
      mientras  no(nul(S1))  hacer
         X : = PRIMERO (S1) ; S1 : = RESTO (S1) ; S2 : = S2 + sal(esq(f (X))))
      fmientras

Esq ( ∂ { f } )
   ent  X : t1 , sal  y : t2
   inst      y : = sal (esq(f(X)))

Esq  hom  { c,f } )
   ent  S : seq  <t >  sal   d:dom   var  X:t ,   d1:dom
   inst      Si  nul(S) entonces   d : = c
             Sino
               X : = PRIMERO (S) ; S : = RESTO (S) ;
               d1 : = sal(esq(hom(S))) ; d : = sal (esq(f(X,d1)))
             fsi
```

3. METODOLOGIA PROPUESTA

La metodología consta de los pasos siguientes:

a. Análisis del problema y elaboración de especificaciones informales.

b. Estudio de las especificaciones informales para determinar:
   - Los Tipos de Datos Abstractos
   - Los operadores importantes

c. Especificar algebráicamente los Tipos de Datos Abstractos, utilizando la notación dada en la Sección I.

d. Especificar funcionalmente los operadores importantes utilizando los constructores funcionales dados en la Sección I.

e. Escribir los operadores en forma funcional, utilizando las formas funcionales asociadas a los constructores y optimizarlos utilizando las reglas de transformación dadas en la tabla II.

f. Elaborar una interfase algoritmica abstracta con los Tipos de Datos Abstractos, que permita la conexión entre los Tipos de Datos Abstractos y el programa algoritmico que los utiliza.

g. Transformar las especificaciones de los operadores en programas algoritmicos, a través de los esquemas de programación de los constructores funcionales dados en la sección II y de la interfase algoritmica abastracta elaborado en el paso 6 de la metodología.

4. CONCLUSIONES

Hemos presentado una metodología de construcción de programas que permite en una forma sistemática elaborar programas confiables, centrando todo su interés en escribir todo el problema una primera vez, a través de un lenguaje sin ambiguedades como lo es el de especificación funcional, lo cual permite visualizar la arquitectura del programa completo. De esta forma se da la idea de todo un proyecto, sus dificultades, sus divisiones lógicas, etc.

La escritura completa de un proyecto en una forma precisa antes de su programación permite hacer modificaciones, estructuraciones, descomposiciones, etc., a nivel de especificaciones.

Los puntos f y g de la metodología permiten de una manera fácil pasar de las especificaciones a una representación algoritmica.

5. BIBLIOGRAFIA

(BACK 78): J. BACKUS
Can programming be liberated from the Von Neuman Style? A functional Style and its Algebra of Programs. CACM Vol. 21, N8 1978.

(BERT 82a) D. BERT
Software Components Construction, in Tool and Notions for Program Construction. Ed. D. Neal. Cambridge University Press. Pag. 347-376.

BERT 82b) D. BERT
Generic Programming: A tool for designing universal operators. Aplication to program Algebra. RR 336 Lab. IMAG Grenoble 1982.

BERT 83) D. BERT
Manual de reférence de LPG. RR 458 Lab. IMAG Grenoble 1983.

(GUT 78a) J.V. GUTTAG; E. HOROWITZ; D.R. MUSSER
Abstract Datas Types and Software validation. CACM Vol. 21 n 12 1978.

(GUT 78b) J.V. GUTTAG; J.J. HORNING
The Algebraics Specification of Abstract Data Types. Acta Informática 10 1978

(GOG 78) J.A. GOGUEN; J.W. THATCHER; E.G. WAGNER
An Initial Algebra Aproach to the Specification, Correctness
and Implementation of Abstract Data Types. Current Trends in
Programming Methodology. Vol. 4, Data Structuring Prentice Hall 1978

SODA - AN INTERACTIVE SOFTWARE DESIGN TOOL

A.M.DE A. PRICE*

SUMARIO

Este trabalho apresenta uma descrição do Software Design Assistant (SODA), uma ferramenta automática conversacional para auxiliar projetistas de software no desenvolvimento de novas aplicações. SODA aceita a especificação gradativa de software fornecendo procedimentos para a verificação de consistência e integridade das especificações, e permitindo consultas ao software em projeto . SODA fornece, também, análises e documentação do projeto, auxilio para sua própria utilização e meios para integrar as fases de projeto e implementação de software.

ABSTRACT

This work introduces the Software Design Assistant (SODA), an interactive automatic tool for helping software developers in the design of new computer applications. SODA supports incremental design providing procedures for checking the consistency and completeness of specifications and for accepting enquiry on the system being designed. It provides design analysis and documentation, helping support and means for integrating the design process with the implementation phase.

* Engenheira Química (UFRGS,1972), Mestre em Informática (PUC-RJ, 1976), Doctor in Philosophy (University of Sussex, U.K., 1984); engenharia de Software: projeto, estimativa de custo e medidas de complexidade de software; Pós-Graduação em Ciência da Computação, Av. Osvaldo Aranha,99 Porto Alegre - RS.

1. Introduction

It is well-known that the cost of software is now the most expensive element of a computer-based system. While hardware costs have continue to drop over the last decade, the cost of software has been rising and now can reach the figure of 85 percent of the total computer system expenditure [Wasserman 82].

These is a need to control the rising costs of software by improving both the process of software production and the quality of the resulting software. A reasonable solution to control software rising costs, over a short term is to make software personnel more effective by providing them with techniques and automatic tools for software development.

Modern software development tools are claimed to increase both the quality of the product and developer productivity, and to enhance job satisfaction for the tool user [Riddle 80].

Although there exist a number of effective software tools,there is a shortage of tools to assist with requirements specification, design and testing [Wasserman 82]. According to Boehm, the cost of system analysis and design account for about 40 percent of the software development costs. Additionally, errors in the design phase which are not detected until testing commences can be very expensive to correct [Sommerville 82].

The motivation for the development of SODA [Price 84b] originated from the abore considerations.

2. The Software Design Assistant

The Software Design Assistant (SODA) is an interactive software design tool providing the following capabilities:

- supporting of incremental software design;
- consistency and completeness checking of design specifications;
- a library mechanism for storing software constructs which can be instantiated as components of the software being designed;
- supporting the design of parallel processing systems;
- supporting enquiry on the system being designed;
- providing design analysis and documentation;
- providing means for integrating the design process with the implementation phase;
- providing helping support, explaining SODA behaviour to the user at each stage of its duty.

SODA enforces a design methodology providing mechanisms to

ensure that the design consistency is maintained across levels of detail. It primarily supports the top-down approach to modular decomposition with flexibility to accept the refinement of lower level modules before completing the specification of higher level ones. It is oriented towards functional specification but also promotes design with data abstractions. SODA perceives a software system as a set of hierarchically structured modules, which can be either sequential (programs, subroutines of functions) or parallel processing (processes) segments. Inter-module data communication is accomplished by parameters and global data is restricted to external files for achieving module independence and information hiding.

SODA supports the design specification of sequential and concurrent systems. Both approaches to process cooperation followed by Ada and Concurrent Pascal can be described by DSL, the SODA design language. DSL provides means for specifying communication and synchronization of processes, including statements such as the REQUEST/ACCEPT operation for process communication and SELECT for describing non-determinism. Mutual exclusion and delaying of monitor operations can be described by the statement SELECT provided with guarded conditions.

3. DSL - The Design Specification Language

Design specifications are input in a PDL-like language [Caine77] which includes means for describing data and module behaviour. PDL'S have proved to be worthy for representing software behaviour for many reasons (e.g. easy to learn, convenient because of their similarity to programming languages , straightforward path to implementation).

The framework for module specification is illustrated below.

```
MODULE  <name>  : <module type> ;
        PARAMETER { INPUT  } <parameter declaration list> ;
                  { OUTPUT }
        EXTERNAL  { INOUT  } <file identifier list>;
        CALL <list of called operations/subprograms> ;
        TYPE <data type declarations> ;
        DATA <data structure declarations> ;
        BEHAVIOUR <statement> ;
        OPERATION <list of offered operations> ;
        SUBCOMPONENT <identification of subcomponents> ;
END
```

The design specifications for a module do not need to be input in the same order as the one presented by the framework above, neither has the entire framework to be input in one sole session with SODA. The designer can input some of the declarations at one time and complete the specifications in later sessions.

SODA includes an interactive SDL processor that provides procedures for analysing the syntax and checking the consistency and completeness of design specifications against the Design Specs Base (DSB). The major checking procedures involve searching for missing definitions and ensuring appropriate use of the defined entities. Inconsistent specifications are discarded but incomplete ones are stored with a special sign to agree with the top-down design approach which accepts references to entities that will be defined later. At each interaction with the system, besides entering new information the designer can also update the information already stored in the DSB. Updating includes commands for appending and deleting parameters, data definitions, behaviour statements, and so on.

DSL is a strongly typed language. It provides a few standard types, but enable its users to define their own data and module types.DSL also accepts parameterized abastract data types which enables different classes of data abstractions to be defined by a single data type definition. It provides a type EITHER which permits the union of any data types and can be used for specifying recursive types. A type EMPTY is provided for specifying empty objects. The combination of types EITHER and EMPTY permits the definition of dynamic data structures without making use of pointer mechanisms [Price 84a]. An example of a type definition for a stack data structure follows.

```
TYPE Stack:  EITHER EMPTY
             OR     RECORD
                      Cell: Celltype;
                      Bottom: Stack;
                    END;
```

The DSL user is also permitted to define modular types. This feature is particularly useful when a modular type is defined as an ALGORITHM in the SODA Library so it can be instantiated as a component of any software project.

DSL includes the procedural specification of abstract data types. There are some advantages of using a procedural language for specifying abstract data types (e.g. semantic descriptions are easy to write, understand, and implement because of their similarity to conventional programming). Abstract data types are declared inDSL as being of type ADT

and are basically composed of two parts: (1) DATATYPE which introduces the type that characterizes the class of data objects defined by the ADT, and (2) a set of operations (subroutines/functions) that can be performed on the objects. A type definition in DSL for the binary search tree abstraction is shown below.

```
TYPE BinaryTree: ADT;

        PARAMETER
            INPUT  RootType  : TYPE ;

        DATATYPE =
            EITHER EMPTY
            OR     RECORD
                       Root:  RootType ;
                       Left:  BinaryTree  (RootType) ;
                       Right: BinaryTree  (RootType) ;
                   END;
        OPERATION

        Search : FUNCTION   LOGICAL ;
            PARAMETER
                INPUT Tree : BinaryTree  (RootType) ;
                      Label: RootType ;

            BEHAVIOUR

                if EXIST (Tree)
                then if Label = Tree.Root
                     then return True
                     else if Label < Tree.Root
                          then Search (Tree.left,Label)
                          else Search (Tree.Right,Label)
                     else return False;

       Insert : SUBROUTINE ;

            PARAMETER
                INPUT  Label : RootType ;
                INOUT  Tree  : BinaryTree  (RootType);

            BEHAVIOUR

                if EXIST (Tree)
                then if Label < Tree.Root
                        then Insert  (Label, Tree.Left)
                        else Insert  (Label, Tree.Right)
```

```
        else begin
                call RISE  (Tree) ;
                set Tree. Root = Label
             end;
END ;
```

DSL includes the following categories of statements: conditional branch (IF-THEN-ELSE and CASE), iteration (WHILE-DO, REPEAT-UNTIL, FOR, LOOP), compound statement (BEGIN-END), assignment, input/output and subroutine call/return. The semantics of the sequential statements is similar to that defined for the block structured programming languages.The process synchronization statements are REQUEST, ACCEPT, SELECT and ACTIVATE. Besides the above categories, any English sentence may be used as a valid DSL statement.

As SODA follows the top-down design approach, calls to subprograms that have not been defined yet are accepted but a warning message is sent to the user. Also, the declaration of entities whose types have not been defined are accepted. Whenever an entity definition is input to the design base the undefined entries in the base are analysed to see if they match with the new definition.

4. Integration with the Implementation Phase

Since SODA intends to provide an integration with the further phases of the software development process, the DSL processor makes an attempt to analyse fully the behaviour statement of software components. If the parsing succeeds the analysed expression is stored in a structured form which is more easily translated to a high level programming language than the design language. If either there are operands that have not been declared, or any incompatibility exists then the processor stores the whole expression as a string of characters.

The integration of SODA designs with the implementation phase can be realized by translating the design specifications stored in the Design Specs Base to a programming language. Specifications are stored in the DSB as structured information rather than text. The internal representation is defined in terms of software elements and control structures and so it is independent of the design language syntax. The idea was to represent internally the control structures proposed by the structures programming methodology so that the design could be translated easily to the popular block structured programming languages. The statements that do not follow DSL'S syntax are output as comments and left to be refined by the programmer.

5. SODA Library

SODA provides a library mechanism for storing abstract data types and algorithms that can be instantiated as components into the software being designed. The types being introduced in the Library may make reference to any of the types already defined. Thus, it is possible to define more complex types by extension of the ones already defined. Whenever a designer specifies an instantiation, the construct is copied from the Library and incorporated into the software being designed. The Library can be searched either automatically (whenever instantiations of the Library types are specified) or manually (the designer can ask SODA either to show the Library directory, a specific abstraction or to get a full copy of the Libraty itself).

6. SODA outputs

SODA provides the capability for asking the data base for information such as which modules call a certain module, or which modules use a given table of data. Besides producing a formated listing of the entire software design (or parts of it), SODA outputs a number of analyses that intend to warn the designer of possible design problem areas. All documents can be obtained either on the screen or on hard copy.

The flow matrices, whose purpose is to show the control and data flows of a software design, constitute one of the SODA analyses. The Control Flow Matrix depicts the calling relations between modules of a software system. The purpose of the Data Flow Matrix is to show the relationship between modules and files/data bases. Therefore, looking at this matrix the designer can see which modules have access to a particular file or data base.

SODA can produce a complete cross-reference listing containing all symbols either defined or referenced in the design project. The Data Dictionary of a software project constitutes a listing of all data definitions specified at the highest level of the software design. The software symbol table, a listing of all symbols defined/referenced in the design project can also be obtained. An example of a formated listing of a design specification, followed by its cross-reference, symbol table and control flow matrices is presented in Appendix A.

SODA includes an automatic design quality metric which can provide immediate feedback to the designer. The design evaluation method has been proposed by Yin and Winchester [Yin 79] and is based on McCabe's program complexity measure [McCabe 76]. Examining the measures of

design complexity the designer can identify levels at the design structure that are error-phone and with help from other SODA analyses he/she can identify the modules that are causing the high complexity and restructure the software accordingly. An example of the design quality metric is shown in Appendix B.

6. Conclusion

A Pascal implementation of a SODA prototype on a VAX-11 computer system has been developed. Most of the SODA features described previously have been implemented by the prototype.

SODA has most requisite characteristics to be a successful software tool [Nassi 80];

- easy to learn and use; SODA's command language is small and fairly simple. Its design language requires no training for people familiar with conventional programming languages;
- good performance; the SODA prototype has shown very good response time;
- easily modifiable; the prototype has been developed according to the principles of modularity, information hiding and data abstraction, which are recognized as successful approaches for the construction of software that is easy to understand and maintain;
- meets a clear need; the design process is known to be the most critical phase of the software development process, incurring heavy costs on the final product. SODA is addressed to the software designer and his problem, providing functions for validating design specifications, producing design documentation and giving design analysis feedback.

The SODA prototype can also be used for educational purposes as a software design laboratory. Software engineering students can make use of it when developing their case study applications as well as for implementing new algorithms of software cost estimation, design analysis and design quality evaluation.

7. References

[Boehm 80] B.W. Boehm, Software Engineering - As it is, in Software Engineering, Freeman and Lewis (eds.), 1980.

[Caine 77] S.H. Caine and E.K. Gordon, PDL - A tool for Software Design, in Tutorial on Software Design Techniques, P. Freeman and A.I. Wasserman (eds.), IEE Inc., 1977, pp. 168-173.

[McCabe 76] T.J. MacCabe, A Complexity Measure, IEEE Transactions on Software Engineering , December 1976, pp. 308-320.

[Nassi 80] I. Nassi, A Critical Look at the Process of Tool Development: An Industrial Perspective, in Software Development Tools, W.E. Riddle and R.E. Fairley (eds.) Springer Verlag, 1980, pp. 40-51.

[Price 84 a] A.M.de A. Price, Defining Dynamic Variables and Abstract Data Types in Pascal, ACM SIGPLAN Notices, February 1984.

[Price 84b] A.M.de A. Price, SODA - An Interactive Design Tool, D. Phil Dissertation, University of Sussex, U.K., December 1984.

[Riddle 80] W.E. Riddle and R.E. Fairley, Software Development Tools, Springer-Verlag, 1980.

[Sommerville 82] I. Sommerville, Software Engineering, Addison-Wesley Publ. Co., London, 1982.

[Wasserman 82] A.I. Wasserman, The Future of Programming, Comm. of the ACM, vol. 25, March 1982, pp. 196-205.

[Yin 79] B.H. Yin and J. W. Winchester, Software Design Quality Metrics System, Second International Conference on Mathematical Modelling, July 1979.

```
-------------------------------------------------------------
MODULE SODA : PROJECT ;

CALL
   Show <<TO BE DEFINED>>, Parser, Update <<TO BE DEFINED>> ;

TYPE
   inBuffer : ADT ;

      DATATYPE = RECORD
                    line : STRING ;
                    index : NUMBER ;
                 END ;

      OPERATION

         Getchar : SUBROUTINE ;

            PARAMETER
            OUTPUT
               symbol : CHAR ;
            INOUT
               buffer : inBuffer ;

            DATA
               Terminal : FILE OF STRING ;

            BEHAVIOUR
 10                BEGIN
 20                   IF end_of_buffer  <<TO BE DEFINED>> THEN
 30                      BEGIN
 40                         READ inputline  <<TO BE DEFINED>> FROM Termi
 50                         initiate index  <<TO BE DEFINED>>;
                         END ;
 60                   get symbol pointed by index  <<TO BE DEFINED>>;
                   END ;
   END ;

DATA
   Terminal : FILE OF STRING ;
   command : STRING ;
   inputBuffer : inBuffer ;

BEHAVIOUR
 10   REPEAT
 20      READ command FROM Terminal ;
 30      IF command = "input" THEN
 40         CALL Parser  ELSE
 50         IF command = "update" THEN
 60            CALL Update  ELSE
 70            IF command = "display" THEN
 80               CALL Show ;
      UNTIL command = "exit" ;

SUBCOMPONENT
   Parser : SUBROUTINE ;
-------------------------------------------------------------
MODULE Parser : SUBROUTINE ;

CALL
   Scanner ;
```

```
SUBCOMPONENT
   Scanner : SUBROUTINE ;
-------------------------------------------------------------
MODULE Scanner : SUBROUTINE ;

PARAMETER
OUTPUT
   token : STRING ;
   code : NUMBER ;

CALL
   inBuffer.Getchar ;

DATA
   blank : CHAR ;
   symbol : CHAR ;
   letter : SET OF CHAR ;
   digit : SET OF CHAR ;
   delimiter : SET OF CHAR ;

BEHAVIOUR
 10   BEGIN
 20      CALL inBuffer .Getchar ;
 30      WHILE symbol = blank DO
 40         CALL inBuffer .Getchar ;
 50      IF symbol in digit THEN
 60         BEGIN
 70            SET code = 2;
 80            WHILE symbol in digit DO
 90               BEGIN
100                  append symbol TO token  <<TO BE DEFINED>>;
110                  CALL inBuffer .Getchar ;
                  END ;
             END ;
120      IF symbol in letter THEN
130         BEGIN
140            SET code = 1;
150            WHILE symbol in ( letter + digit ) DO
160               BEGIN
170                  append symbol TO token  <<TO BE DEFINED>>;
180                  CALL inBuffer .Getchar ;
                  END ;
             END ;
       END ;
-------------------------------------------------------------
```

CROSS REFERENCE LIST

SYMBOL	CLASS	DEFINED AT	REFERENCED AT
Update	UNDEFINED		SODA
Show	UNDEFINED		SODA
Terminal	DATA	SODA	SODA
command	DATA	SODA	SODA
inBuffer	TYPE	SODA	Scanner SODA
Getchar	OPERATION	inBuffer	Scanner inBuffer
line	RECORDFIELD	inBuffer	NO REFERENCES
index	RECORDFIELD	inBuffer	NO REFERENCES
inputBuffer	DATA	SODA	NO REFERENCES
Update	UNDEFINED		SODA
Show	UNDEFINED		SODA
Parser	MODULE	SODA	SODA
Scanner	MODULE	Parser	Parser
token	PARAMETER	Scanner	NO REFERENCES
code	PARAMETER	Scanner	Scanner
symbol	DATA	Scanner	Scanner
blank	DATA	Scanner	Scanner
letter	DATA	Scanner	Scanner
digit	DATA	Scanner	Scanner
deliniter	DATA	Scanner	NO REFERENCES

SYMBOL TABLE

SCOPE SODA	CLASS	TYPE
Update	UNDEFINED	
Show	UNDEFINED	
Terminal	DATA	FILE
command	DATA	STRING
inBuffer	TYPE	ADT
Getchar	OPERATION	SUBROUTINE
line	RECORDFIELD	STRING
index	RECORDFIELD	NUMBER
inputBuffer	DATA	inBuffer
Update	UNDEFINED	
Show	UNDEFINED	
Parser	MODULE	SUBROUTINE

SCOPE Parser	CLASS	TYPE
Scanner	MODULE	SUBROUTINE

SCOPE Scanner	CLASS	TYPE
token	PARAMETER	STRING
code	PARAMETER	NUMBER
symbol	DATA	CHAR
blank	DATA	CHAR
letter	DATA	SET
digit	DATA	SET
deliniter	DATA	SET

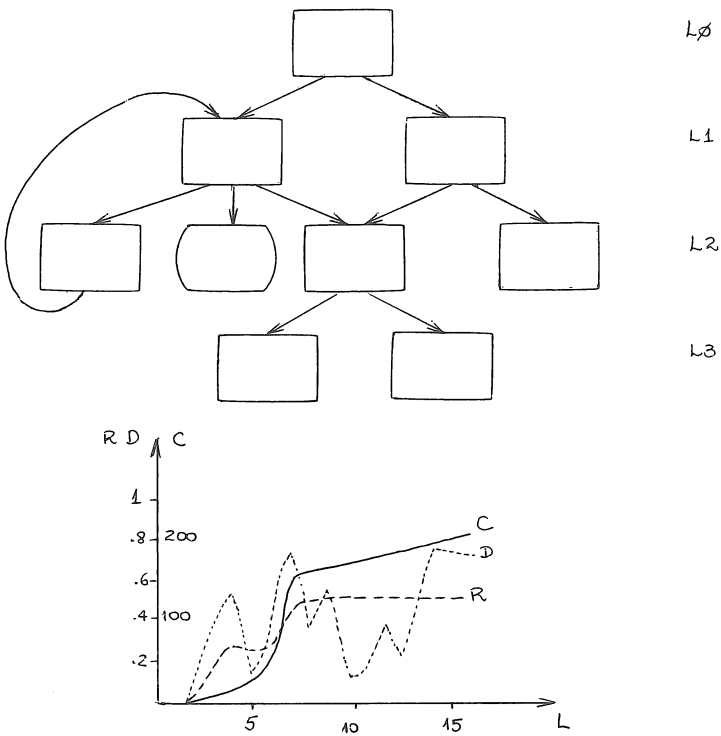
```
-------------------------------------------------------------
          CONTROL FLOW MATRIX "ORDER  1"
-------------------------------------------------------------
+--+ SJDA
|  |
+--+--+ Show
|->|  |
+--+--+--+ Parser
|->|  |  |
+--+--+--+--+ Update
|->|  |  |  |
+--+--+--+--+--+ Scanner
|  |  |->|  |  |
+--+--+--+--+--+--+ inBuffer.Getchar
|  |  |  |  |->|  |
+--+--+--+--+--+--+
-------------------------------------------------------------
          CONTROL FLOW MATRIX "ORDER  2"
-------------------------------------------------------------
+--+ SJDA
|  |
+--+--+ Show
|  |  |
+--+--+--+ Parser
|  |  |  |
+--+--+--+--+ Update
|  |  |  |  |
+--+--+--+--+--+ Scanner
|->|  |  |  |  |
+--+--+--+--+--+--+ inBuffer.Getchar
|  |  |->|  |  |  |
+--+--+--+--+--+--+
-------------------------------------------------------------
          CONTROL FLOW MATRIX "ORDER  3"
-------------------------------------------------------------
+--+ SJDA
|  |
+--+--+ Show
|  |  |
+--+--+--+ Parser
|  |  |  |
+--+--+--+--+ Update
|  |  |  |  |
+--+--+--+--+--+ Scanner
|  |  |  |  |  |
+--+--+--+--+--+--+ inBuffer.Getchar
|->|  |  |  |  |  |
+--+--+--+--+--+--+
-------------------------------------------------------------
          CONTROL FLOW MATRIX "COMPLETE"
-------------------------------------------------------------
+--+ SJDA
|  |
+--+--+ Show
|->|  |
+--+--+--+ Parser
|->|  |  |
+--+--+--+--+ Update
|->|  |  |  |
+--+--+--+--+--+ Scanner
|->|  |->|  |  |
+--+--+--+--+--+--+ inBuffer.Getchar
|->|  |->|  |->|  |
+--+--+--+--+--+--+
```

Appendix B

The design complexity evaluation method works on design structure charts as the one shown below. The rectangular and convex boxes represent processing modules and data base tables respectively, while the arrows represent control and data transfer. The method includes three functions which are computed for each level of a design tree structure:

- C, the cyclomatic number, which measures the network complexity. A sharp increase of C from one level to the next indicates an extreme complexity increase in the software system;
- R and D, which measures the tree-impurity of software system; i.e., the deviation of a design tree structure from an ordinary graph. R measures the tree-impurity of level i against level Ø, and D measures that of level i against level i-1.

The graph below shows the complexity measures computed for a large project (involving approximately 15 levels of design structure hierarchy).



A LINGUAGEM CONSTRUCTOR PARA AMBIENTES COMPUTACIONAIS

P. KEGLEVICH DE BUZIN *

SUMARIO

Neste trabalho são apresentadas algumas das características mais marcantes da linguagem CONSTRUCTOR, com alguns poucos exemplos. Tal linguagem foi projetetada tendo vista quatro tipos de ambientes computacionais: seqüencial, concorrente, parelelo e distribuído.

ABSTRACT

This paper presents some of the most important features of the CONSTRUCTOR programming language, with some few examples. Such language was designed aiming four kinds of computational environments: sequential, concurrent. parellel and distributed.

* MsC. Ciência da Computação (UFRGS, 1984); Sistemas Distribuídos, controle de processos e sistemas em temp-real, engenharia de software, projeto de linguagens de programação. UFRGS-PGCC, Av. Osvaldo Aranha, 99 - Porto Alegre, RS - CEP 90000.

1. INTRODUÇÃO

O fornecimento de recursos computacionais através de uma linguagem de alto nível confere uma enorme versatilidade e confiabilidade a um sistema, provendo um meio eficiente de descrever as computações feitas pela máquina. Além disso, constitui-se em um meio efetivo para descrever e divulgar computações entre pessoas.

A seguir será feita uma descrição suscinta da linguagem de alto nível CONSTRUCTOR projetada recentemente para suporte e desenvolvimento de sistemas em ambientes computacionais. Maiores detalhes, e as considerações que nortearam o projeto desta linguagem, os quais são omitidos neste trabalho, podem ser encontrados em |KEG 84a, KEG 84b|.

2. A LINGUAGEM CONSTRUCTOR

Todos os comandos da linguagem seguem a filosofia dos comandos guardados, com a única diferença que a conferência das guardas e feita seqüencialmente. A idéia a que esta mesma estrutura seja mantida pelo código virtual gerado pelo compilador.

O projeto da linguagem não pode deixar de preocupar-se com a implementação. Assim, o conjunto completo da linguagem foi subdividido em oito subconjunto de complexidade gradativa de modo que é possível haver oito versões intermediárias da linguagem, até chegar a versão final. Este esquema é similar ao adotado na implementação da linguagem SP/K. A linguagem CONSTRUCTOR e definida de modo que a partir da segunda versão é possível escrever o compilador da linguagem na própria linguagem.

A concepção da linguagem foi precedida da elaboração de um modelo para ambientes computacionais, dentro do qual existem quatro tipos de ambientes computacionais: seqüencial, concorrente, parelelo e distribuído. A seguir é mostrado como a linguagem descreve cada ambiente.

2.1 AMBIENTE SEQUENCIAL

A parte algoritmica seqüencial foi bastante influenciada pelas linguagens PASCAL e EDISON |HAN 81|, possuindo porém um menor número de comandos. A declaração de tipos e similar a da linguagem EDISON, pois facilita a compilação. Também foram adicionados alguns aprimoramentos em relação ao PASCAL.

2.1.1 ESTRUTURA DE DADOS

Na definição da estrutura de dados, só é permitida a declaração de tipos novos definidos pelo usuário, que não sejam padrão do sistema. Além disso, o tipo ARRAY e definido pelo usuário especificando, ao invés do número de dimensões que possui, o tipo de indexação usado e o tipo do elemento básico que o compõe. Os limites são estabelecidos na declara-

ção de variáveis. Portanto o tipo ARRAY funciona como uma espécie de tipo parametrizado. Desta maneira, mantendo as mesmas regras de definição de parâmetros do PASCAL, é possível fazer com que as procedures e functions sejam aplicadas a arrays de qualquer limite ou tamanho.

Exemplos:

```
TYPE  typeline(INT)CHAR;
VAR   line|1..80|:typeline;
      impline|1..132|:typeline;
PROCEDURE manipula (VAR buf:typeline; VAL tam:INT);
VAR  l:INT;
  .
  .
ENDPROC;
      .
      .
manipula(line,80);
manipula(impline,132);
```

Também é permitido que os limites de variáveis tipo ARRAY declaradas dentro de procedures ou functions possam ser especificados por expressões contendo constantes e parâmetros passados "by-value".

Exemplos:

```
PROCEDURE proc (VAL a,b:INT);
VAR line |a..b+2| : typeline;
```

2.1.2 COMANDOS SEQÜENCIAIS

Os comandos seqüenciais podem ser subdivididos em comandos simples e comandos compostos. Os comandos simples são os de atribuição, de avaliação de construtores (EVAL), chamada de procedure e o comando de saída de comando estruturado (OUT). Os comandos estruturados são o de alternação (IF) e o de repetição (WHILE). Cada um destes comandos pode conter várias guardas seguidas de comandos. Cada um destes grupos de comandos só é executado quando a guarda estiver aberta, ou seja, for verdadeira.

2.1.3 PROCEDURE E FUNCTIONS

Quando houver uma procedure passada como parâmetro, os parâmetros da mesma devem ser também especificados na declaração de parâmetros da procedure.

Exemplos:

```
PROCEDURE bisect(FUNCTION f(VAL x:FLOAT):FLOAT;VAL l,h:FLOAT);
```

É permitido o aninhamento de procedures e functions, e ainda é possível a declaração particionada de procedures. Isto é feito especificando um cabeçalho que antecede uma posterior declaração completa da procedure.

Como o modelo em que se baseia a linguagem permite a utilização de memória compartilhada como meio de comunicação, considerou-se interessante dar liberdade a linguagem para descrever procedures reentráveis. Nos contextos de ambientes paralelos e concorrentes , isto pode permitir uma boa eficiência de uso de memória, além de fornecer recursos adicionais.

Outra facilidade oferecida pela linguagem e a de execução de rotinas em biblioteca. Isto implicitamente implementa o conceito de overlay sobre a pilha de dados do processo.

Exemplo:

LIB PROCEDURE Spascal(VAL source, arqlist,code:indentifier)|SPASCAL|;

2.2 AMBIENTE CONCORRENTE

A linguagem CONSTRUCTOR e fortemente baseada na linguagem SR |AND 82| na parte concorrente. Basicamente foram incluídos alguns aprimoramentos em relação a linguagem SR. principalmente quanto a uma maior disciplina no acesso a recursos de baixo nível, e o fornecimento de estruturas que facilitem uma programação mais clara e uma compilação mais eficiente.

2.2.1 COMANDOS CONCORRENTES

O conceito de operação permite a linguagem descrever mecanismos das mais variadas características semânticas de uma maneira concisa e clara. De acordo com o modelo de ambiente computacional, as operações consistem em mensagens com uma determinada semântica associada ao envio e ao recebimento das mesmas. Além disso, a liberdade de descrever a semântica das operacões confere a linguagem a habilidade de implementar eficientemente qualquer mecanismo de sincronização desejado. Os processos, também descritos pela linguagem, são vistos como um recurso disponível para programar a semântica do mecanismo das operações.

Basicamente os comandos concorrentes atuam sobre o ambiente como um todo e sobre operações. O comando que atua sobre todo o ambiente e o ABORT, que interrompe a execução de todos os processos do ambiente. Os comandos sobre operações se subdividem entre aqueles que definem operações e aqueles que invocam operações. As operações são definidas pelo comando SELECT, dentro de um processo. Os parâmetros formais de uma operação são os mesmos de uma procedure.

O comando SELECT é composto de uma ou várias guardas. Cada guarda e especificada pelo nome da operação definida pela mesma, e opcionalmente, por uma parte de sincronização e outra de escalonamento. Após cada guarda, segue-se o conjunto de comandos da operação. Portanto, um mesmo comando SELECT pode definir várias operações. Quando um processo executa um comando SELECT, fica suspenso até que uma das guardas deste comando seja aberta.

Quando há uma parte de sincronização, esta é especificada pela cláusula WHEN seguida de uma expressão booleana. Assim uma guarda e considerada aberta quando existe uma invocação pendente para a operação que define, e a condição especificada pela expressão booleana é verdadeira.

No caso de haver a parte de escalonamento, esta é especificada pela cláusula BY seguida de uma ordenção no atendimento das invocações, quando existe mais de uma invocação pendente. Aquela invocação para a qual o valor da expressão aritmética for menor, será a primeira a ser atendida. Naturalmente os parâmetros da operação podem entrar nesta expressão aritmética.

Exemplos:

```
SELECT read(RES v:item;i:INT) DO
       CALL  Startread;
       chave, v:=true, database |i|;
       SEND Endread;
ALSO Write(VAL v:item;i=ITEM) WHEN chave DO
       CALL  Startwrite;
       database|i|:=v;
       SEND Endwrite;
ANDSL;
SELECT do-io(c,...) BY ABS(c-position) DO
    position:=c;
ENDSL;
```

As operações podem ser invocadas por comandos CALL ou SEND, seguido do nome da operação com os parâmetros reais da chamada. No caso do SEND, o processo que invoca a operação não espera que esta termine a sua execução para continuar. Já no caso do comando CALL, o processo aguarda a confirmação de que a execução da operação terminou, antes de continuar.

Assim o comando CALL correspondende a uma chamada síncrona de operação, e o comando SEND correspondente a uma chamada assíncrona de operação.

Com o comando CONC é possível invocar várias operações no modo CALL paralelamente. Este comando é seguido de uma lista de chamada de operações com seus parâmetros reais. A execução do processo só irá continuar após todas as operações desta lista terem terminado as suas execuções.

Exemplos:

```
SEND put(c);
CALL startwrite;
CALL read(c);
CONC starwrite, read(c) ENDC;
```

2.2.2 ESTRUTURA DE DADOS

Para dar liberdade ao programador de controlar dinamicamente o contexto de operações que acessa, é implementado pela linguagem o conceito de capability de operações (CAP). Isto é interessante para implementar privacidade em sistemas operacionais.

Exemplo:

```
TYPE CAP io(input(RES buffer:line);
          output(VAL buffer:line)):
VAR leitesc:io; buf:line;


 leitesc.input:=read;          "read write são operações"
 leitesc.output:=write;        "pre-definidas"
CALL leitesc.input(buf);
CALL leitesc.output(buf);
```

2.2.3 MÓDULOS

O conceito de módulos do CONSTRUCTOR e uma extensão da idéia de Wirth |WIR 77|, onde os módulos isolavam partes do programa dependentes do tempo e de sincronização. No caso do CONSTRUCTOR, os módulos isolam características funcionais e sintáticas diferentes. Isto permite ao compilador detectar alguns erros de concepção de programa em tempo de compilação, aumentando a segurança dos programas escritos na linguagem.

Na linguagem CONSTRUCTOR existem três tipos de módulos: tipo OBJECT, que especifica os processos do ambiente, tipo INTHARD, que especifica uma interface de hardware, e tipo INTCOMP, que define um interface computacional. Cada um destes módulos apresentam construções sintáticas particulares, conforme a função a que se destinam. Não serão descritas as características sintáticas de cada módulo neste trabalho, apenas serão apresentados alguns exemplos explicativos.

O módulo tipo INTHARD descreve como o ambiente computacional irá "enchergar" a parte do hardware utilizada pelo sistema. Neste módulo são descritas as posições físicas de memória, as portas de I/O da máquina, as interrupções, e a forma de acesso pelo programa a estes recursos de baixo nível. A seguir é apresentado um exemplo de interface hardware de uma console descrito em CONSTRUCTOR:

```
INTHARD  intcons;
COUPLE console;
TYPE ARRAY registers(INT)BOOL;
VAR tks|0..7|:registers AT @177530; "reg. status keyboard"
    tps|0..7|:registers AT @177540; "reg. status printer"
    tkb:CHAR AT @17750; "Keyboard data register"
    tpb:CHAR AT @17751; "printer data register"
ACCESS Kint(tkb) INTERRUPT PORT @60;
       print INTERRUPT PORT @64;
ACTUATE startk; tsk|6|:=true; ENDACT;
ACTUATE startp; tps|6|:=true; ENDACT;
ACTUATE sckk; tks|6|:=false; ENDACT;
ACTUATE ackp; tps 6 :=false; ENDACT;
ACTUATE put(VAL c:CHAR); tpb:=c; ENDACT;
ENDHARD;
```

O módulo tipo INTCOMP (ou MSINTCOMP no caso de interface mestre) especifica a maneira como dois ambientes computacionais (programas) se intercomunicam. Especificam as operações importadas e exportadas, as operações importadas e exportadas, as eventuais exceções que possam ser atendidas, e a forma de controle da intercomunicação. A seguir é apresentado um exemplo do interface aplicado para a carga de programas por parte de um ambiente mestre (normalmente um sistema operacional), e o uso de primitivas por parte de um ambiente escravo (que pode também conter vários processos). Assim este interface funciona como um prefixo para o ambiente escravo, de uma maneira similar ao sistema SOLO de Brinch Hansen |HAN 77|.

No ambiente mestre o interface teria a forma:

```
INTCOMP programa;
IMPORT read(REF c:CHAR) /CALL/;
       write(VAL c:CHAR) /SEND/;
       etc...
COUPLE loader;
    .
    .
ENDCOMP;
```

No ambiente escravo teria a forma:

```
MSINTCOMP prefix;
EXPORT read(REF c:CHAR) /CALL/;
       write(VAL c:CHAR) /SEND/;
       etc...
ENDCOMP;
```

Um módulo OBJECT define um ou mais processos concorrentes que podem se comunicar entre si ou com o ambiente externo ao módulo por meio de operações. As operações que são visíveis ao ambiente externo ao módulo são descritas no início do mesmo, com regras de escopo bem estabelecidas.

Um módulo OBJECT pode conter outros módulo OBJECTs e até definir outros ambientes compu

tacionais. Exemplificar tal estrutura seria muito extenso para este trabalho, isto pode ser encontrado em |KEG 84a, KEG 84b|.

2.3 AMBIENTES PARALELOS

Em ambientes paralelos, a parte algoritmica e a mesma que a do ambiente concorrente. O problema novo que surge é a necessidade descrever a distribuição dos módulos na arquitetura da máquina hospedeira, que nestes casos deve possuir mais de um processador. Assim os elementos do ambiente devem ser identificados (ELEMENT), a conexão entre tais elementos (topologia) deve ser descrita (CONNECT), e deve ser de designada a localização dos módulos lógicos (LOCALIZE). No ambiente paralelo os elementos podem ser processadores (PROCESSOR) ou memória compartilhada (SHAREDMEM).

Exemplo:

```
CONSTRUCT HOST(parelelo);
ELEMENT PROCESSOR P1(1);
        PROCESSOR P2(2);
        SHAREDMEM mem(@17000);
CONNECT (P1,mem); (P2,mem);
LOCALIZE (modi,P1); (mod2,P2); (Compr,mem);
```

2.4 AMBIENTES DISTRIBUÍDOS

No caso de ambientes distribuídos, a situação é similar a dos ambientes paralelos, a única diferença na descrição e que os elementos são nodos. A indicação de que a descrição a seguir é de uma computação em ambiente distribuído, e feita pela cláusula NET.

Exemplo:

```
CONSTRUCT NET(rede);
ELEMENT NODE N1(1);
        NODE N2(2);
        NODE N3(3);
CONNECT (N1,N2); (N2,N3); (N3,N1);   "topologia em anel"
LOCALIZE (modi,N1); (mod2,N2); (mod3,N3);
```

3. CONCLUSÃO

Em todo o projeto houve a preocupação com a implementação da linguagem. Assim, por exemplo, foi especificado um símbolo diferente para indicação de fim para cada tipo de estrutura diferente, pois isto facilita a recuperação de erros no parsing. Além disso, o tratamento de entrada e saída pode ser totalmente feito em CONSTRUCTOR, o que irá aliviar grandemente a implementação do kernel do sistema. Também foi eliminada a possibilidade de criação e destruição dinâmica de processos dentro de um programa CONSTRUCTOR, isto confere uma complexidade desnecessária ao sistema, e com vantagens duvidosas.

O projeto do sistema CONSTRUCTOR e totalmente baseado em conceitos comuns e correntes em ciência da computação, e introduz um novo estilo de programação. De uma maneira simplificada, uma descrição de uma computação em CONSTRUCTOR e equivalente a uma descrição de um sistema computacional físico (computador) mais a descrição de um programa (algoritmo mais estrutura de dados).

BIBLIOGRAFIA

|AND 82| ANDREWS, G.R. The distributed programming language SR - mechanisms, design, and implementation. Software - Practice & Experience, 12(8):719-53, 1982.

|HAN 77| HANSEN, P.B. The architecture of concurrent programs. Englewood Cliffs, Prentice-Hall, 1977.

|HAN 81| _____. Edison - a multiprocessor language. Software - Practice & Experience, 11(4):325-61, 1981.

|KEG 84a| KEGLEVICH DE BUZIN, P.F.W. Uma abordagem para a concepção e implementação de sistemas distribuídos. Porto Alegre, PGCC da UFRGS, 1984. (Dissertação de mestrado).

|KEG 84b| _____. Constructor: projeto de uma ferramenta para descrição de computações. Prêmio SERPRO de monografia âmbito nacional, primeiro lugar. Nov. 1984.

|WIR 77| WIRTH, N. A programming language for modular multipromming. Software - Practice & Experience, 7:3-35, 1977.

ARNDT VON STAA*

SUMARIO

E descrito um critério para a seleção de casos teste por cobertura. O critério parte de diagramas de organização modular do programa (diagramas estruturados), criando uma expressão regular descrevendo todos os caminhos existentes neste programa. De posse desta expressão regular são determinados os caminhos a serem executados, um para cada caso teste. Os caminhos são selecionados utilizando-se um critério que procura simular indução matemática para o caso de repetições. Ao final são discutidas extensões e algumas limitações do critério.

ABSTRACT

An open box test data selection criterion is described. The criterion departs from a graphical representation of the program's structure. A regular expression is derived from this structure, describing all paths of the program to be tested. Once the regular expression has been formed, the paths to be tested are identified. The identification process attempts to simulate mathematical induction.

* PhD em Ciência da Computação. Areas de interesse: engenharia de software, ambientes de desenvolvimento de software, controle de qualidade de software. E professor associado no Departamento de Infomática da Pontifícia Universidade Católica do Rio de Janeiro

Este artigo recebeu apoio de: FINEP contrato: 62.84.0416.00 e de FIPEC contrato: 1468-0

1. Introdução

Teste de programas continua a ser um problema relevante, apesar dos resultados já alcançados na àrea de prova formal de programas. Testar um programa não é submetê-lo a uns tantos testes e "ver no que dá". Testar é, antes de tudo, a seleção racional de experimentos (casos testes) e a condução controlada destes experimentos (testes) [Myers79]. O teste sistemático de um programa visa identificar as deficiências deste programa. Caso não identifique deficiências, o teste sistemático deve ser capaz de atestar o grau de confiabilidade do programa. É claro que, no caso geral, testes não conseguirão assegurar a correção absoluta de um programa. Tampouco a prova formal tem-se mostrado capaz deste intento [Gerhard77]. Admitem os menos radicais que a virtude está a meio caminho. Ou seja, deve-se balancear o esforço investido em testes e em prova de correção. O resultado disto será um programa confiável desenvolvido com economia de recursos [Howden80].

Um dos critérios de seleção de casos teste è a cobertura de caminhos [Beizer83]. Este critério de cobertura de caminhos gera, a partir da organização interna do programa, casos teste que cubram todos os caminhos de um determinado conjunto de caminhos definidos no programa. É amplamente conhecido que o conjunto de todos os caminhos de determinado programa pode ser infinito. Precisa-se, pois, selecionar casos teste de modo que:

i- o conjunto de caminhos a serem examinados seja finito e relativamente pequeno, de modo que o custo do teste não exceda o necessário para assegurar o alcance do nível de qualidade desejado

ii- o conjunto de caminhos a serem examinados seja uma seleção válida. Uma seleção è válida [Goodenough77] se, existindo falha no programa, será gerado pelo menos um caso teste que exercite esta falha e acuse um erro, i.e. uma diferença entre o resultado esperado e o resultado computado.

Critérios de seleção de casos teste caixa aberta, em particular a cobertura de caminhos, têm-se baseado no fluxograma do programa. Na inexistência de geradores automáticos de casos teste, isto è no mínimo um inconveniente, uma vez que fluxogramas tendem a gerar somente trabalho adicional, pouco ou nada contribuindo para um melhor entendimento do programa. Em outras palavras, um bom critério de seleção de casos teste deve auxiliar o controlador de qualidade a esmiuçar o programa, auxiliando-o e motivando-o a argumentar formalmente a correção do programa sendo examinado.

O critério de seleção apresentado neste artigo parte de diagramas de organização modular de programas. Estes diagramas são o resultado do projeto do programa. Ou seja, a base para a seleção de casos teste é o projeto do programa e não o seu fluxograma. O primeiro passo é converter diagramas de organização modular em expressões algébricas (seção 2). Para programas não recursivos, a expressão algébrica é uma expressão regular. A linguagem regular correspondente a esta expressão algébrica é o conjunto de todos os caminhos possíveis no programa. De posse da expressão algébrica é realizada uma seleção criteriosa visando determinar o conjunto de expressões de caminhos a serem percorridos pelos casos teste (seção 3). Cada um destes caminho corresponde a um caso teste. Ao criar caminhos envolvendo ciclos procura-se simular a indução matemática como um meio de tornar finito o número de caminhos a testar. Finalmente, na seção 4 são examinados alguns problemas pendentes, e é feita uma avaliação do critério de seleção.

2. Geração da expressão algébrica

Como já foi mencionado, o ponto de partida do critério de seleção de casos teste é o diagrama de estrutura organizacional do programa (diagrama estruturado). Adotamos aqui uma linguagem de representação semelhante á descrita em [Staa83]. A esta linguagem de representação são adicionadas as regras:

i- cada bloco de ativação repetitiva contém uma única repetição. Esta ativa um bloco de inicialização e um único bloco executor das instâncias desta repetição

ii- cada bloco de ativação seletiva contém uma única seleção

A figura 1 ilustra resumidamente a linguagem de representação gráfica utilizada neste artigo. A figura 2 ilustra o uso desta linguagem de representação em um projeto detalhado de programa.

Cabe observar que a presente linguagem de representação é capaz de representar qualquer programa procedural. Além disso é fácil mostrar que as linguagens de representação utilizadas por [Yourdon79] e por [Jackson75] são conversíveis para a linguagem de representação adotada neste artigo.

Para efeito deste artigo impomos as seguintes restrições adicionais:

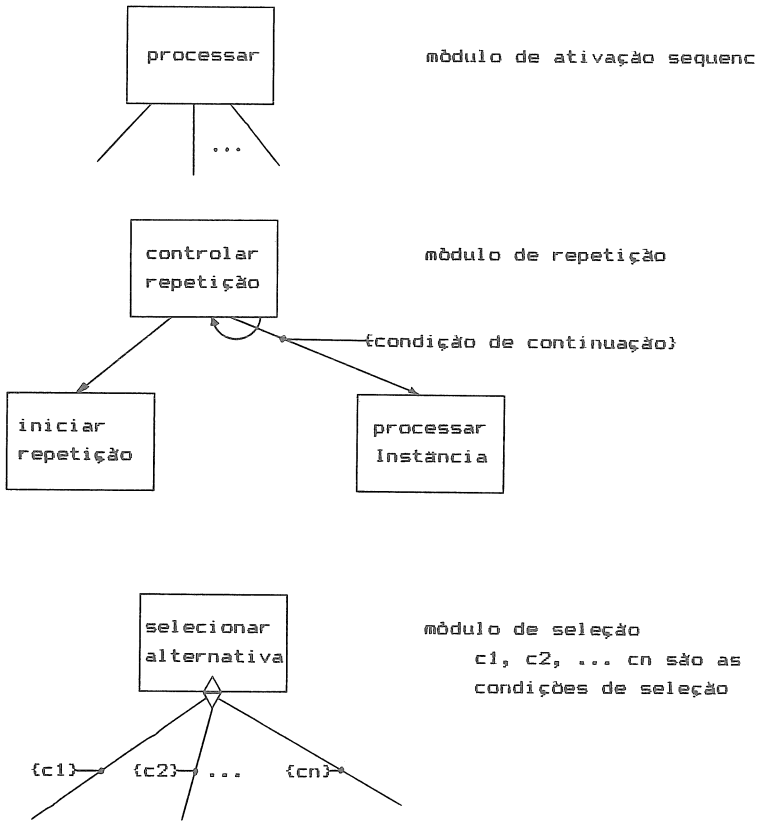


Figura 1. Descrição da linguagem de representação

i- não é permitida a recursão. Ou seja, o grafo representativo do projeto do programa é acíclico. No entanto, não é necessário que seja puramente arborescente (i.e. pode ter "fan in" maior do que 1).

ii- não é permitida a suspensão da execução (aborto) em módulos quaisquer. Ou seja, entrada e saída são sempre via o módulo raiz do programa.

Estas restrições não são demasiadamente fortes. Primeiro, uma grande gama de programas interessantes é formada por programas não recursivos. Na seção 4 será comentado um possível caminho para eliminar

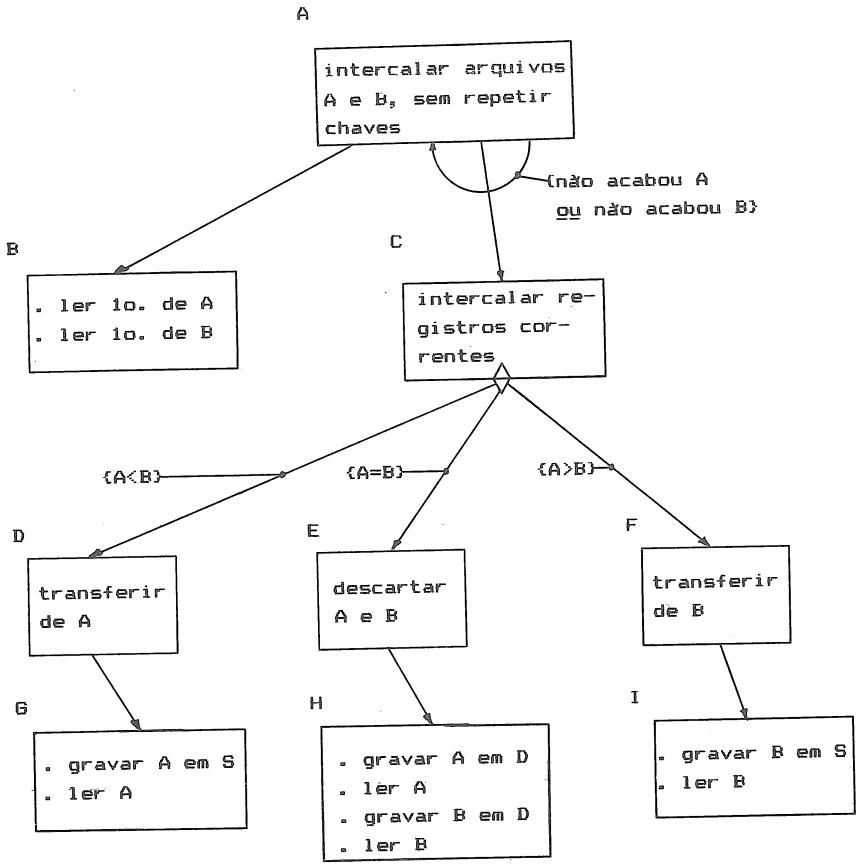


Figura 2. Exemplo de projeto detalhado. Programa de intercalação de arquivos sem repetir chave. O arquivo resultado è S. O arquivo de erros - chaves duplicadas - è D. Entradas são arquivos A e B, assumidos corretos.

esta restrição. E, segundo, a interdição de supensão em qualquer lugar pode ser resolvida através do uso de um sinal indicador de condição de continuação. Esta regra è equivalente á regra bàsica da programação estruturada, ou seja, cada bloco tem uma ùnica entrada e uma ùnica saida.

Antes de iniciar a geração da expressão algèbrica do grafo

correspondente á estrutura modular do programa, rotula-se este grafo. No exemplo da figura 2 utilizamos letras para rotular os blocos. Após executa-se o algoritmo descrito na figura 3. A figura 4 ilustra alguns passos da execução deste algoritmo utilizando o programa da figura 2.

```
gerar expressão( raiz )

   caso tipo do bloco raiz

      raiz nula:
         faz nada

      bloco simples:
         imprimir rótulo do bloco raiz
         para cada aresta do bloco raiz repetir
            gerar expressão( bloco ativado por[ aresta ])
         fim repetir

      bloco condicional:
         imprimir rótulo do bloco raiz
         imprimir "("
         gerar expressão( bloco ativado por[ primeira aresta])
         para cada aresta de bloco raiz,
                        a partir da segunda inclusive repetir
            imprimir "¦"
            gerar expressão( bloco ativado por[ aresta ])
         fim repetir
         se condição è opcional
         então
             imprimir "¦"
             imprimir "ø"          /* nulo */
         fim se
         imprimir ")"

      bloco repetitivo:
         imprimir rótulo do bloco raiz
         gerar expressão( inicializar raiz)
         imprimir "["
         gerar expressão( repetir raiz )
         imprimir "]*"

   fim caso
fim gerar expressão
```

Figura 3. Algoritmo recursivo para a geração da expressão algèbrica

ao chegar em "intercalar registro"

A B [C

ao terminar o algoritmo

A B [C (D G ¦ E H ¦ F I)]*

após incorporar o arrasto (seção 3)

A B [2 C (D G ¦ E H ¦ F I)]*

Figura 4. Ilustração da execução do algoritmo

3. Geração de expressões de caminho

Uma vez de posse da expressão algébrica que descreve todos os caminhos existentes no programa sendo examinado, passa-se a selecionar o conjunto de caminhos que deverão ser percorridos durante os testes.

A primeira providência a ser tomada é determinar um critério para a escolha de caminhos envolvendo repetições, uma vez que estas tendem a tornar infinito o número de caminhos a serem examinados.

Repetições dependem de uma variável ou de um estado de controle (exemplo de estado de controle: registro lido). Em adição variáveis e/ou estados são modificados a cada ciclo. Em princípio pode-se imaginar a existência de funções, uma para cada variável ou estado modificado durante o ciclo, na forma:

VAL[i] := F(VAL1[i-1], VAL2[i-2], ... , VALn[i-k])

onde i é o índice da instância atual, i-1 o índice da instância anterior, etc., e i-k o índice da k-ésima instância anterior, tal que k seja mínimo e F dependa de VAL[i-k]. Chama-se k de arrasto. O arrasto da repetição é o maior arrasto encontrável nesta repetição. A seguir apresentamos alguns exemplos de valores de arrasto:

i- ao incializar um vetor atribuindo 0 a todos os elementos, o arrasto é 0, uma vez que esta inicialização independe de valores calculados em instâncias anteriores.

ii- ao calcular o somatório dos valores contidos em um vetor, o arrasto é 1, uma vez que a soma da instância atual depende da

soma parcial até a instância anterior

iii- ao calcular um novo elemento da série de Fibonacci, o arrasto é 2, uma vez que a fórmula de cálculo do i-ésimo elemento desta série é a[i]:=a[i-1]+a[i-2].

iv- no exemplo da figura 2 o arrasto é 2, uma vez que a variável de controle - número de registros lidos - é avançada de dois no caso de serem descartados registros em virtude da igualdade das chaves.

Intuitivamente pode-se dizer que o arrasto da repetição é igual ao número mínimo de ciclos necessários para que todas as variáveis e/ou estados modificados durante a repetição passem a depender exclusivamente de valores computados em ciclos anteriores. Ou seja, o arrasto da repetição é o número mínimo de ciclos necessário para esta repetição atingir o "steady state".

A determinação do arrasto de cada uma das repetições depende do entendimento do programa. Em geral o arrasto não é determinável por meios mecânicos simples. Sendo assim, é função do projetista determinar o arrasto de cada repetição contida no programa. Com o intuito de tornar mecânica a seleção do caminhos a serem percorridos durante os testes, sugere-se que os valores dos arrastos sejam introduzidos na expressão algébrica logo após ao "[" correspondente ao início da repetição. A figura 4 ilustra a expressão algébrica assim modificada.

A seguir formulamos o critério de seleção para repetições:

i- os casos {i = 0, 1, ..., arrasto} são casos especiais, e precisam ser testados um a um. São casos especiais, uma vez que correspondem a um estado transiente de inicialização do ciclo.

ii- em adição á inicialização é necessário, também, efetuar um teste para um número n genérico de iterações. É óbvio que desejamos o menor n possível com o intuito de minimizar o volume de dados e, consequentemente, minimizar o custo da execução do teste. Uma possível solução é utilizar um valor de n (número de ciclos a executar) tal que n > arrasto. Justifica-se este critério, observando que o arrasto é o menor número de iterações a ser realizado assegurando que todos os valores modificados durante a repetição tenham sido computados estritamente a partir de valores já computados em iterações anteriores desta repetição.

O critério acima procura simular a indução matemática. Ou seja, os arrasto primeiros testes formam a base da indução e o teste envolvendo

```
{ buffer vazio ao iniciar }

   gerar caminho( posição )

      caso token em posição

         fim de expressão algébrica:
            imprimir buffer

         rótulo de módulo:
            adicionar rótulo ao final do buffer
            gerar caminho( próxima posição )
            retirar rótulo do final do buffer

         "(":
            para cada  elemento i da alternativa  repetir
               computar posição inicial da i-ésima alternativa
               gerar caminho( posição calculada )
            fim repetir
            gerar caminho( posição após ")" )

         ")":
            /* não deveria ser encontrado */

         "[":
            para  i = 0, 1, ..., arrasto+1  repetir
               para  j de 1 até i  repetir
                  gerar caminho( posição após indicador de arrasto
               fim repetir
            fim repetir
            gerar caminho( posição após "]*" )

         "]*":
            /* faz nada, provoca retorno da recursão */

      fim caso

   fim gerar caminho
```

Figura 5. Algoritmo recursivo para a geração de caminhos visando cobertura completa.

n>arrasto ciclos simula o número "genérico" de repetições (hipótese de indução) ao qual é adicionado um novo ciclo "genérico" (passo de

indução). É óbvio que, sendo uma simulação utilizando um número definido não genérico, a validade no caso geral desta simulação não pode ser assegurada. Consequentemente, será possível identificar diversos exemplos de programas provadamente errados, para os quais o critério de seleção é incapaz de determinar um conjunto válido de casos teste (i.e. nenhum caso teste consegue determinar a existência do erro). Apesar desta potencial falibilidade do critério de seleção, não foi possível, até agora, encontrar um tal contra exemplo.

Existem diversas possíveis estratégias para selecionar os caminhos. Os casos extremos são:

i- cobertura completa, selecionam-se os caminhos para todas as possíveis combinações de alternativas e/ou número de ciclos a executar em cada uma das repetições.

ii- cobertura simples, selecionam-se suficientes caminhos de modo que cada alternativa e/ou repetição seja completamente testada, sem, no entanto, exigir o teste de combinações destas condições.

Deve estar claro que a cobertura completa gera o número máximo de caminhos cobrindo o programa, considerando o critério de seleção para repetições. Deve estar claro, ainda, que a cobertura simples gera o número mínimo de casos teste cobrindo todo o programa e satisfazendo o critério de seleção para repetições.

Neste artigo vamos nos restringir á cobertura completa. A figura 5 mostra o algoritmo utilizado para gerar todos os caminhos. A figura 6 ilustra o resultado da aplicação deste algoritmo ao problema da intercalação de arquivos sem repetição de chaves.

4. Avaliação do critério de seleção

Foram conduzidos diversos experimentos utilizando este critério de seleção de casos teste. Os resultados têm sido muito animadores, uma vez que o critério de seleção tem-se mostrado confiável, ou seja, efetivamente apontou erros sempre que havia falhas no programa sendo testado. No entanto, ainda são necessários muitos experimentos para poder-se determinar o grau de confiabilidade do presente critério de seleção.

O critério de seleção tem-se mostrado animador também sob outro ponto de vista, que é o de forçar o entendimento do programa por parte do examinador. Ou seja, a aplicação deste critério de seleção de casos teste induz o examinador a conduzir uma argumentação formal da correção do programa. Apesar de possivelmente incompleta, a argumentação tende a ser suficiente para assegurar ao examinador o entendimento preciso do

A B	A e B vazios
A B C D G	A cnt 1, B vazio cnt : contém
A B C E H	A cnt 1, B cnt 1, A=B
A B C F I	A vazio, B cnt 1
A B C D G C D G	A cnt 2, B vazio
A B C D G C E H	A cnt 2, B cnt 1, B=A[2]
A B C D G C F I	A cnt 1, B cnt 1, A<B
A B C E H C D G	A cnt 2, B cnt 1, B=A[1]
A B C E H C D H	A cnt 2, B cnt 2, A[1]=B[1], A[2]=B[2]
A B C E H C F I	A cnt 1, B cnt 2, A=B[1]
A B C F I C D G	A cnt 1, B cnt 1, A>B
A B C F I C E H	A cnt 1, B cnt 2, A=B[2]
A B C F I C F I	A vazio, B cnt 2
A B C D G C D G C D G	A cnt 3, B vazio
A B C D G C D G C E H	.
A B C D G C D G C F I	.
A B C D G C E H C D G	.
. . .	
A B C F I C F I C F I	A vazio, B cnt 3

Ao todo 3**3 + 3**2 + 3**1 + 3**0 = 40 casos

Figura 6. Caminhos a serem testados no programa da figura 2.

programa. A combinação de argumentação com testes tem-se mostrado altamente eficiente e eficaz como método de avaliação da qualidade do programa. É eficiente uma vez que o custo da argumentação dirigida não é demasiado, tampouco o é o custo de teste. É eficaz uma vez que o resultado é um programa altamente confiável.

O critério de seleção tem outra virtude que é a de ser hierárquico. Ou seja, é possível gerar casos teste para organizações modulares onde cada módulo é relativamente complexo, mas onde a estrutura é relativamente simples, i.e. contém poucos componentes. Desta forma o método se presta, em princípio, para apoiar a geração de testes de integração de programas complexos.

O maior defeito do critério de seleção é a sua tendência á explosão, i.e. geração de um número muito grande de casos teste. Para organizações modulares um pouco mais complexas (número elevado de repetições e/ou seleções) isto pode tornar inviável a aplicação econômica do critério de seleção. É verdade, entretanto, que o número de dados por caso teste tende a ser pequeno e, consequentemente, o custo de

cada execução de teste tende a ser também pequeno. Porém o grande número de casos teste pode tornar proibitivo o custo total.

Presentemente estão sendo estudados diversos caminhos para conter a explosão de casos teste:

i- identificar meios de eliminar caminhos a serem percorridos utilizando para isto informação especifica relativa ás condições de repetição e/ou de seleção. Estas condições podem ser associadas ao diagrama de organização modular tal como ilustrado na figura 2. A aplicação deste mètodo de redução leva a uma execução simbòlica parcial (envolvendo as condições).

ii- verificar se è possìvel identificar de modo mecânico a independência de porções dos programa. Isto também resultarà numa redução de caminhos a testar uma vez que è desnecessàrio testarem-se combinações de condições no caso de regiões independentes.

iii- desenvolver armaduras de teste [Staa83] capazes de aplicarem os testes e confrontar os resultados esperados com os obtidos. Tais armaduras permitirão a execução automàtica dos diversos testes sem a necessidade de intervenção por parte do examinador.

Finalmente, está sendo examinado como estender o presente critèrio de seleção de modo a suportar recursão. Um caminho imaginado, mas ainda não examinado, è o de aplicar á recursão a mesma idèia que a utilizada para o caso de repetições. Isto levarà a diversas expressões regulares, uma para cada condição de percurso da recursão. A razão para se tentar este caminho è o fato da recursão levar a linguagens livres de contexto, ao invès de regulares, complicando assim, no gaso geral, a geração de expressões algèbricas capazes de sintetizar todos os caminhos possìveis no programa.

Referências bibliográficas

[Beizer83] Beizer, B.
Software Testing Techniques; New York; Van Nostrand Reinhold; 1983; 290 pags

[Gerhard77] Gerhard, S.L.; Yelowitz, L.
"Observations of the Fallibility in Applications of Modern Programming Methodologies"; in [Miller77]; 1977; pags 86-98

[Goodenough77] Goodenough, J.B.; Gerhard, S.L.
"Toward a Theory of Test Data Selection"; in [Miller77]; 1977; pags 68-85

[Howden80] Howden, W.E.
"Functional Program Testing"; in IEEE Transactions on Software Engineering SE-6(2); New York; março 1980; pags 162-169

[Jackson75] Jackson, M.A.
Principles of Program Design; New York; Academic Press; 1975

[Miller77] Miller, E. ed.
Tutorial on Program Testing Techniques; New York; IEEE ref. EHO 130-5; 1977

[Myers79] Myers, G.J.
The Art of Software Testing; New York; John Wiley; 1979; 177 pags

[Staa83] Staa, A.v.
Engenharia de Programas; Rio de Janeiro; Livros Técnicos e Científicos; 1983; 286 pags

[Yourdon79] Yourdon, E.; Constantine, L.L.
Structured Design: Fundamentals of a Discipline of Computer Program and System Design; Englewood Cliffs; Prentice-Hall; 1979

J. C. B. Leite* e H. S. Castro**

SUMARIO

O trabalho trata de uma arquitetura de um microcomputador para aplicações em tempo real sob a ótica da tolerância a falhas. Um sistema duplex é proposto aonde a identificação da unidade falha é feita através da comparação de figuras de mérito dos canais. O conceito de dissimilaridade é usado de forma a evitar que falhas em modo comum devido a erros de projeto causem a perda do sistema.

ABSTRACT

This paper investigates the architecture of a microcomputer for real-time applications from a fault-tolerance viewpoint. A duplex system is proposed where the identification of a faulty unit is done by comparison between merit indicators in each channel. The dissimilarity concept is applied with the objective that design errors do not cause a system crash.

* Engenheiro Eletrônico (PUC/RJ, 1974), Mestre em Engenharia Elétrica (PUC/RJ, 1977), Ph.D. (UMIST, 1983); sistemas distribuídos, computação tolerante a falhas; Professor do Depto. de Eng. Elétrica da PUC/RJ, Rua Marquês de São Vicente, 225 - CEP 22453 - Rio de Janeiro, RJ.

** Engenheiro Eletricista (UFCe, 1982), Mestrando em Engenharia Elétrica (PUC/RJ); sistemas distribuídos, computação tolerante a falhas, redes locais; Depto.de Eng. Elétrica da PUC/RJ, Rua Marquês de São Vicente, 225 - CEP 22453 - Rio de Janeiro, RJ.

1. Introdução

O desenvolvimento da indústria de computação e do própio parque industrial brasileiro permite prever que, a médio prazo, técnicas de tolerância a falhas em sistemas de computação venham a ser empregadas em alguns segmentos da indústria nacional.

Esta necessidade já hoje manifesta-se em setores de tecnologia de ponta, como no caso da indústria espacial brasileira. Um exemplo concreto, é o caso do desenvolvimento do computador de bordo do satélite brasileiro, que vem sendo feito no INPE/CNPq, em São José dos Campos.

Como todo e qualquer equipamento, o computador está sujeito a falhas, as quais por sinal devem ser consideradas como elementos normais na vida de qualquer sistema. Como tais, ou seja, como eventos esperados, a elas devemos associar um processo de tratamento que permita a recuperação do sistema e a continuidade de sua operação. Claramente, nem todas as aplicações requerem um alto grau de segurança mas, em certos casos, a interrupção do funcionamento pode ser desastrosa do ponto de vista econômico e em outros pode estar colocando vidas humanas em perigo.

No que se segue, é descrita a implementação da primeira fase do projeto de um microcomputador tolerante a falhas para aplicações em tempo real. O mesmo está em fase de testes devendo em breve o hardware ser declarado operacional. O desenvolvimento de um programa monitor mais elaborado e o estudo e implementação de técnicas de recuperação de erro por programa fazem parte de um outro estudo a ser iniciado.

2. Algumas Definições

Entendemos por um sistema de computação tolerante a falhas aquele que, sem intervenção manual, tem a capacidade de tratar falhas operacionais e de projeto e manter seu comportamento como definido em sua especificação. Duas técnicas complementares são normalmente usadas para a obtenção de sistemas confiáveis [AVIZ 78, DEPL 81]. No primeiro caso, investe-se no uso de melhores componentes, melhor montagem, etc., o que implica em um melhor MTBF. A segunda técnica, faz uso da introdução de redundância para o aumento da confiabilidade. Neste trabalho, estamos mais interessados no segundo aspecto, que envolve maiores considerações conceituais e uso de técnicas específicas para a implementação desta redundância.

Pode-se dividir em quatro fases as estratégias básicas usadas com qualquer tipo de redundância protetiva [ANDE 81]: deteção de erro, avaliação dos danos, recuperação do

erro e tratamento da falha. Em algumas destas fases, mecanismos podem ser implementados para a sua execução, como no caso dos chamados blocos de recuperação. Em outras, devido por exemplo a problemas de custo, simplesmente os problemas são resolvidos a priori. Pode-se citar aqui a fase de avaliação dos danos aonde, por simplificação, as falhas podem ser assumidas ter determinadas caracterfsticas e os danos por elas causados são então analisados e avaliados tão somente na fase de projeto.

Falhas de um modo geral podem ser classificadas em previstas e imprevistas. As primeiras são aquelas que podem ser esperadas com um certo grau de precisão sobre suas causas e consequências. Por exemplo, elas podem ocorrer devido ao envelhecimento de componentes, interferência elétrica, etc.. No segundo grupo estão os modos de falha desconhecidos e interações imprevistas entre o sistema e o seu meio ambiente. Além dessas, ainda no segundo grupo, pode-se acrescentar as falhas de projeto, tanto de hardware quanto de software. Finalmente, um erro é definido como o resultado produzido a partir de entradas incorretas ao circuito em questão causadas pela ação de uma falha.

3.Objetivos do Projeto

Em sua forma mais geral, a meta deste trabalho é o projeto e implementação de um microcomputador de baixo custo, tolerante a falhas, para uso em tempo real. A partir daf, uma série de pontos podem ser levantados, que são a seguir detalhados. Exemplos de sistemas deste tipo podem ser encontrados em [KAME 80, PATE 82 , TOY 78].

Primeiramente por motivos de custo e complexidade, optou-se pela realização de uma estrutura em **dois canais**, em detrimento de estruturas TMR ou outras mais elaboradas. Neste caso, o mecanismo básico de deteção de erros consiste na comparação das saídas dos dois canais paralelos, que operam sobre o mesmo conjunto de dados de entrada. E importante notar que a localização de uma falha fica mais difícil, embora esta ainda possa ser realizada com o suporte da análise dos erros e de testes. Se os canais utilizados são indênticos, o que facilitaria a elaboração do software de suporte, problemas surgiriam ao nível das falhas não antecipadas (p.ex., falhas de projeto), que propagar-se-iam da mesma maneira em ambos os canais, impedindo então sua deteção. Desta forma, optou-se pelo uso da chamada redundância dissimilar, que pode fornecer um mecanismo de deteção de erros bem eficiente [MESQ 79, RIAL 80]. Obviamente isto é um fator de complicação , e para minorar o seu impacto ao nível da complexidade, os canais foram projetados com base nos microprocessadores Z80 e 8085. Como é sabido, o conjunto de instruções do 8085 é um subconjunto do conjunto das instruções do Z80, o que pode simplificar a fase inicial de depuração do sistema. E importante ressaltar que cuidados devem ser tomados para que a implantação do software de suporte em cada canal seja feita

com diferentes codificações, e possivelmente diferentes algoritmos. Esta última decisão deve ser pesada em relação ao aumento em custo a ser incorrido.

Outro ponto importante é a exigência de **não existir uma unidade central de gerenciamento**, ou ainda que, no caso da existência de elementos críticos ('hardcore'), que estes tenham confiabilidade bem mais elevada do que um canal básico. Um exemplo de tal elemento crítico é o chamado seletor de saída. Em um sistema duplex, o elemento que faz a comparação entre as saídas dos canais é, essencialmente, uma unidade de gerenciamento. Assim, sua operação deve ser livre de erros, pois uma falha na comparação pode permitir a propagação de uma saída errônea, o que vai contra os objetivos do projeto. De modo a reduzir-se o riscos de falhas e ainda, para evitar a existência de elementos centrais, decidiu-se realizar a comparação por software, pela troca de informações entre os canais. Além disso, os próprios dados enviados ao seletor de saída, bem como a saída do próprio seletor devem ser realimentadas. Vale observar que o procedimento de comparação por software é mais lento do que caso o mesmo fosse implementado em hardware. Isto no entanto não constitue um problema nesta máquina experimental. Finalmente, uma restrição fundamental é que não haja um elemento central de sincronização.

Um fato importante para um microcomputador tolerante a falhas é que o mesmo sempre forneça saída correta, a menos que haja uma indicação de erro. Um sistema com tal característica é chamado de **seguro a falhas.** Assim, além das falhas simples ele deve também ser capaz de detetar ou evitar as chamadas falhas em modo comum. Isto é aqui conseguido pelo uso de canais dissimilares e independentes. Além disso, o software de comunicação entre os canais deve ser livre de bloqueio e um canal não pode ter a permissão de interromper o outro.

Outra característica exigida neste projeto é que a estrutura tenha **capacidade de degradação.** Ou seja, o microcomputador deve ser capaz de detetar falhas permanentes e transientes, tentar sobreviver aos seus efeitos e, caso não seja possível a recuperação do canal atingido, funcionar em modo simplex, gerando um alarme relativo a esta situação. Para que esta capacidade de degradação possa ser implementada é necessário que mecanismos de deteção de erro sejam implantados e que se faça a indicação do canal falho. Os mecanismos de deteção aqui usados são:

- 'time-out' em intercomunicações;
- paridade na memória;
- comparação de entradas, saídas e trocas de mensagem;
- endereçamento fora de limites;
- programas de testes periódicos;
- 'rollback' em programas repetitíveis.

Além disso, estão previstos testes de validade sobre os limites dos valores de entrada e saída, sua taxa de variação, etc.. Com isto, outras situações anormais não detetadas pelos mecanismos referidos podem ser indicadas.

Finalmente, toda redundância empregada deve ser **transparente** ao usuário. Mecanismos devem ser elaborados tais que a implementação da tolerância a falhas possa se dar automáticamente. Assim, além dos diversos detetores, inserção de pontos de recuperação, etc., técnicas como as propostas em [LOQU 85] devem ser investigadas. Isto contudo faz parte de um outro estudo, relativo a implementação do monitor de controle.

4.Sincronização

Em virtude das diferentes caracterfsticas dos processadores usados e com o objetivo de eliminar elementos centrais, os canais desta máquina funcionam de forma independente, cada qual com sua própia base de tempo, sincronizando-se via troca de mensagens.

Duas formas de sincronização estão previstas, uma associada ao início de ciclo de operação e outra quando da troca de informações entre tarefas. Assim, admite-se a existência de um ciclo de operações em tempo real ao início do qual os canais trocam informações e reiniciam suas bases de tempo. Isto é necessário pois como são previstos canais distintos, com requerimentos próprios de relógio, fatalmente há a propagação de uma (pequena) diferença entre as bases de tempo dos canais. Além disso, é também fornecido o suporte para a troca de informações entre tarefas durante um ciclo de operações. O diagrama da figura 1 dá uma indicação do mecanismo de sincronização.

5.Descrição do Sistema

O sistema foi projetado em torno de uma estrutura baseada em dois canais fazendo-se uso do conceito de redundância dissimilar. A sua arquitetura está indicada na figura 2. Cada um dos canais é um microcomputador capaz de, isoladamente, assumir o processo de controle. Assim, ambos terão estrutura semelhante, baseada em uma arquitetura clássica, contendo os mesmos tipos de recursos. Do ponto de vista da implementação, o sistema foi realizado em três placas, sendo uma para cada canal e a terceira para a implementação do seletor de saída e parte da comunicação com o sistema de desenvolvimento MCPUC-80.

Cada canal contém os recursos abaixo relacionados, sendo alguns deles associados ao processo de deteção e recuperação de falhas:

- unidade central de processamento de 8 bits;
- 4k de EPROM;

- 4k x 9 de memória RAM (8 bits mais paridade) e armazenamento de endereço errôneo;
- registros para limites de endereço de execução;
- unidade de comunicação inter-canal;
- unidade de temporização;
- unidade de E/S;
- unidade de comunicação serial com o sistema de desenvolvimento;
- indicadores de status.

O teste de cada parte do sistema é feito ou através de indicadores próprios, como no caso da memória RAM, ou através de temporização, como no caso das comunicações entre canais, ou por comparação, como no caso das saídas ou ainda por testes periódicos. Algumas unidades podem ter mais de um tipo de teste de funcionamento. É importante ressaltar que da eficiência destes testes dependerá o sucesso da operação de deteção e recuperação, ou seja, da **cobertura** desta operação. Este ponto será abordado quando da avaliação da confiabilidade.

Para exame da memória ROM ficaríamos limitados a operações de sequências de teste periódicas, o que pode não ser conveniente dado que a deteção do erro poderia ficar muito distante no tempo do instante de sua ocorrência. Assim, optou-se por um esquema aonde, quando da gravação da ROM é acrescentada uma última palavra de teste sobre todo o conteúdo anterior. Quando o sistema é ligado, o conteúdo desta ROM é transferido para a RAM, a transferência testada para detetar a ocorrência de erros, dando-se então a execução a partir do programa gravado agora em RAM, que é eficientemente testada pelo esquema de paridade.

Quanto ao seletor de saída é implementado usando-se um esquema de multiplexação aonde a seleção de quem comanda a saída é feita independentemente por cada canal, através de uma porta ou-exclusivo. Assim, mesmo que o comando de saída de um canal 'cole' em um certo valor, ainda é possível fazer a escolha correta da saída a partir do outro canal. Esta parte do circuito foi deliberadamente mantida pequena para garantir que sua confiabilidade seja alta quando comparada com a do sistema que a precede. É interessante lembrar que esta parte é um elemento série do ponto de vista de confiabilidade.

Finalmente vale acrescentar que as comunicações inter-canal são redundantes, sendo todas as trocas de mensagem limitadas em tempo. Um protocolo de comunicação pode ser eficientemente implementado fazendo-se uso de portas de E/S paralelas especiais, como por exemplo a Intel 8255.

6.Estimativa da Confiabilidade

O uso de redundância só se justifica no caso aonde a mesma é capaz de fornecer algum grau de proteção. Desta forma, esta proteção deve ser quantificada para haver a garantia de que a mesma é efetiva nas condiçoes de operação e dentro do chamado tempo de missão.

Vários parâmetros podem ser escolhidos como indicadores desta proteção, sua escolha sendo dependente do uso a que se destina o sistema em questão. Neste caso, estamos interessados em um sistema que funcione ininterruptamente por um determinado período de tempo, que não excede 100 horas. Assim, um parâmetro adequado para a quantificação da proteção oferecida pela redundância é a confiabilidade. Esta é definida como a probabilidade de que o sistema funcione corretamente, nas condições especificadas, por um dado período de tempo T, referido como tempo de missão.

Uma das formas mais aceitas e usadas para a estimativa da confiabilidade é a chamada modelagem analítica, da qual faremos uso. É assumido aqui que o processo de chegada das falhas é caracterizado pela lei Poisson e que os tempos associados a ação de recuperação são exponencialmente distribuídos.

Seja então o modelo da figura 3 aonde os números nos estados indicam a quantidade de unidades boas, c é a cobertura da ação de recuperação e os σ_i representam taxas de falha. Aplicando os métodos clássicos de solução para sistemas Markovianos, chegamos ao resultado abaixo:

$$R(t) = p_2(t) + p_1(t)$$

$$p_2(t) = e^{-\sigma_2 t}$$

$$p_1(t) = \begin{cases} c\sigma_2 t e^{-\sigma_1 t} & \text{para } \sigma_1 = \sigma_2 \\ \dfrac{c\sigma_2}{\sigma_2 - \sigma_1}\left[e^{-\sigma_1 t} - e^{-\sigma_2 t}\right] & \text{para } \sigma_i \neq \sigma_2 \end{cases}$$

Para a ação de recuperação podemos usar um modelo baseado em CAST [CONN 77, GEIS 83] (figura 4) aonde d é a probabilidade de deteção da falha, f é a probabilidade de que haja fuga de uma falha transiente (ou seja, a mesma é tomada como permanente), e é a

eficiência da ação de recuperação de falhas permanentes, λ é a taxa de falhas permanentes e μ é a taxa de falhas transientes.

Dado que os tempos envolvidos nos estados DET, TR e PM são muito menores do que os tempos associados aos outros estados, o modelo da figura 4 pode ser reduzido ao da figura 3, aonde σ_2, σ_1 e c são dados por:

$$\sigma_2 = 2(1 - d)(\lambda + \mu) + 2d(\lambda + f\mu)$$

$$\sigma_1 = (\lambda + \mu)$$

$$c = \frac{2de(\lambda + f\mu)}{\sigma_2}$$

Como pode-se notar os σ_i representam as taxas de falha efetivas e c, a cobertura, é a probabilidade de chaveamento correto.

Além dos dois canais, para que o sistema possa efetivamente operar, tem-se o seletor de saída, que neste caso é um elemento série do ponto de vista de confiabilidade. Desta forma a confiabilidade do sistema é dada pelo produto da confiabilidade do seletor pela a confiabilidade da estrutura paralela.

Terfa-se ainda de considerar a fonte de alimentação, mas como o objetivo aqui é observar os ganhos relativos a uma unidade simplex, que também deve ser alimentada, isto não é feito. Assim, os gráficos que se seguem representam tão somente a confiabilidade da estrutura duplex comparada com a de uma estrutura simplex.

Na figura 5 temos um gráfico comparativo entre as estruturas duplex e simplex para o caso aonde a cobertura é de aproximadamente 90%, que pode ser considerado um valor baixo [RENN 84]. As curvas (1,2) e (3,4) representam limites para a confiabilidade das estruturas duplex e simplex, respectivamente, para as variações possíveis na taxa de falhas de um canal. O efeito do seletor de saída na estrutura duplex foi considerado, embora o mesmo seja bem pequeno. Note-se ainda que no caso simplex não foram consideradas as falhas transientes, responsáveis pela maioria das interrupções em um sistema.

7.Conclusão

O sistema apresentado representa parte do esforço desenvolvido no Grupo de Sistemas de Computação do DEE-PUC/RJ na área de computação tolerante a falhas. Certamente o mesmo não é um produto acabado, muito ainda tendo de ser feito. Podemos citar aqui a gerência de reconfiguração e o tratamento de eventos assíncronos. Os autores gostariam de agradecer aos Profs. O. G. Loques Filho e G. F. G. da Silveira pela leitura do texto e suas sugestões de melhoria. Contudo, quaisquer incorreções são única responsabilidade dos autores.

8.Referências

[ANDE 81] T.Anderson e P.A.Lee, "Fault-Tolerance: Principles and Practice", Prentice/Hall International Inc., Londres, 1981.

[AVIZ 78] A.Avizienis, "Fault-Tolerance: The Survival Attribute of Digital Systems", Proc. of IEEE, Vol.66, No.10, Outubro de 1978, p.1109-1125.

[CONN 77] R.Conn et al., "CAST - A Complementary Analytic-Simulative Technique for Modeling Complex Fault-Tolerant Computing Systems", Proc. AIAA Comput. Aerospace Conf., Los Angeles, C.A., Novembro de 1977.

[DEPL 81] P.G.Depledge, "Fault-Tolerant Computer Systems", IEE Proc., Vol.128, Part-A, No.4, Maio de 1981, p.257-272.

[GEIS 83] R.M.Geist e K.S.Trivedi, "Ultrahigh Reliability Prediction for Fault-Tolerant Computer Systems", IEEE Trans. on Computers, Vol.C-32, No.12, Dezembro de 1983, p.1118-1127.

[KAME 80] M.Kameyama e T.Higuchi, "Design of Dependent-Failure-Tolerant Microcomputer System Using Triple-Modular Redundancy", IEEE Trans. on Computers, Vol.C-29, No.2, Fevereiro de 1980, p.202-206.

[LOQU 85] O.G.Loques Filho, "Uma Arquitetura Flexível para Sistemas Distribuídos Tolerantes a Falhas", submetido ao SEMISH 1985.

[MESQ 79] A.C.Mesquita Jr., "Tolerância a Falhas Transientes por Diversificação", Tese de Mestrado, Dep. de Eng. Elétrica, PUC/RJ, Agósto de 1979.

[PATE 82] D.C.Patel, "Dual Dissimilar Processor System for High Reliability", Ph.D. Thesis, Dept. of Electrical Eng. and Electronics, University of Manchester Institute of Science and Technology, Outubro de 1982.

[RENN 84] D.A.Rennels, "Fault-Tolerant Computing - Concepts and Examples", IEEE Trans. on Computers, Vol.C-33, No.12, Dezembro de 1984, p.1116-1129.

[RIAL 80] A.Rial F., "Dissimilar Redundancy in Fault-Tolerant Microcomputer Systems", Ph.D. Thesis, Dept. of Electrical Eng. and Electronics, University of Manchester Institute of Science and Technology, Outubro de 1980.

[TOY 78] W.N.Toy, "Fault-Tolerant Design of Local ESS Processors", Proc. IEEE, Vol.66, No.10, Outubro de 1978, p.1126-1145.

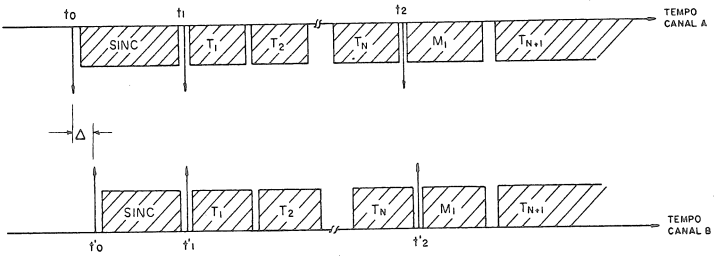


Figura 1 - Sincronização

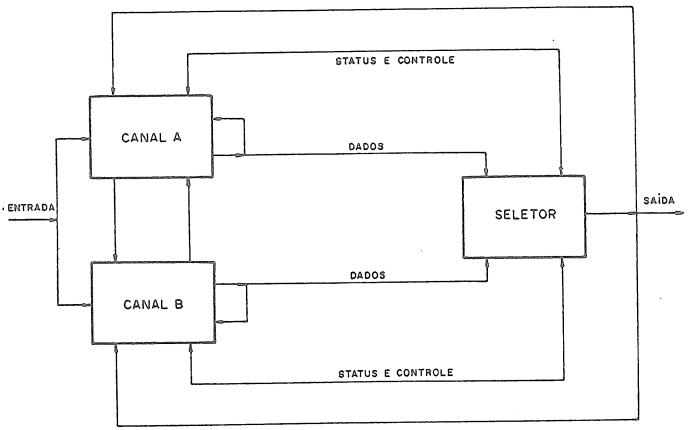


Figura 2 - Arquitetura Básica

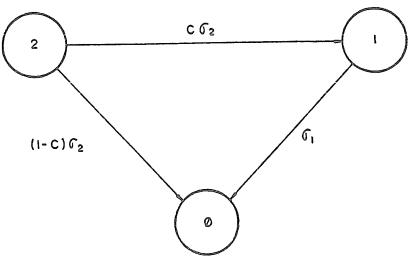


Figura 3 - Modelo de Falhas

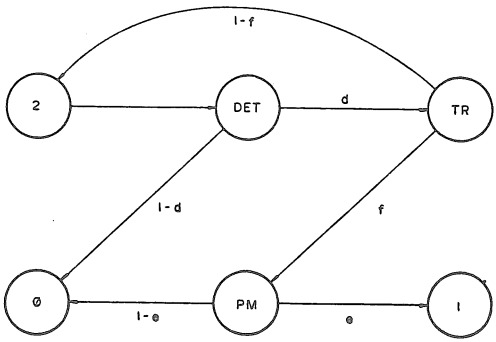
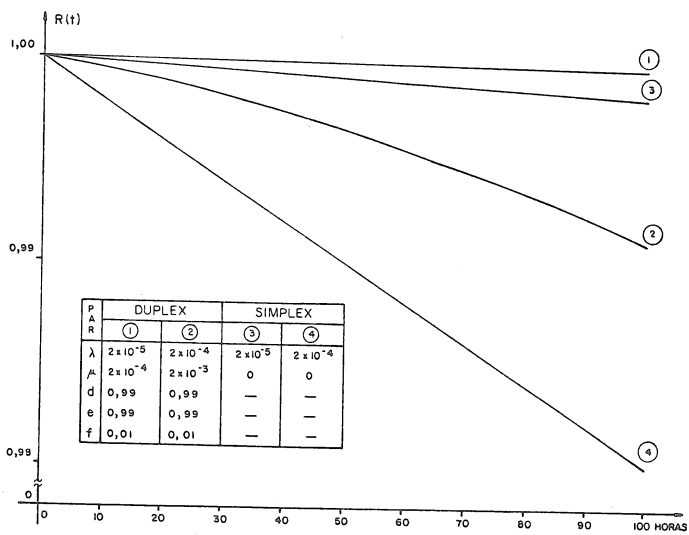


Figura 4 - Modelo de Cobertura



PAR	DUPLEX		SIMPLEX	
	①	②	③	④
λ	2×10^{-5}	2×10^{-4}	2×10^{-5}	2×10^{-4}
μ	2×10^{-4}	2×10^{-3}	0	0
d	0,99	0,99	—	—
e	0,99	0,99	—	—
f	0,01	0,01	—	—

Figura 5 - Confiabilidade: Duplex x Simplex

CIRCUITOS AUTOTESTAVEIS: OPÇAO DE PROJETO DE CIRCUITOS VISANDO O TESTE IMPLICITO OU CONCORRENTE.

I. JANSCH *

SUMARIO

A partir de uma abordagem genérica inicial sobre a testabilidade dos circuitos, estuda-se em detalhe as propriedades de autoteste desses, com vista à detecção de falhas em funcionamento. A evolução da teoria clássica à moderna é analisada e vislumbra-se a conveniência da inclusão dessas propriedades nos circuitos a serem concebidos. Somente as principais definições de base são formuladas - o enfoque predominante é conceitual.

ABSTRACT

Initially, a generic approach introduces the idea of testability. From this one, we study in detail the self-checking properties of the circuits, in view of on-line faults detection. The evolution from the classic theory to the modern one is analyzed, conducting us to see that these properties may be conviniently included in circuits during their design. Only the basic definitions are formulated - this paper is mainly conceptual.

* Engenheira Eletrônica (UFRGS, 1979), Mestre em Ciência da Computação (UFRGS, 1982), Doutora-Engenheira em Microeletrônica (INP de Grenoble, 1985). Endereço: PGCC/UFRGS - Caixa Postal 1501 - 90.000 PORTO ALEGRE - RS.

I. Introdução

Os avanços atingidos pela tecnologia de circuitos integrados têm tornado possível a realização de dispositivos muito complexos em um único "chip" (circuito integrado). Isto provocou uma consequência indesejável: o aumento do tempo de geração de teste. Achar em pouco tempo um conjunto de testes de comprimento razoável que assegure uma cobertura satisfatória de falhas tornou-se um problema de dificuldade crescente à medida que a complexidade do circuito aumenta. Sabe-se que o tempo necessário para a geração automática de testes e a simulação de falhas em um circuito combinacional aumenta com o cubo do seu número de portas lógicas; já em circuitos sequenciais, o tempo de geração de teste pode ser muito maior do que o anterior, dependendo do seu número de estados. Por outro lado, uma baixa cobertura de falhas no teste do chip leva ao decréscimo do rendimento a nível de placa ou sistema.

O custo de localização e reparação de uma falha cresce em um fator da ordem de 10 à medida que se evolui de um chip a uma placa e de uma placa ao nível do sistema - portanto, o teste do chip com alta cobertura de falhas se torna necessária. Mas, referindo-se a um circuito VLSI, isto pode ser impossível de ser obtido em um tempo razoável. E a solução para este problema é incluir testabilidade como restrição de projeto. O resultado desta decisão é a geração de metodologias de projeto visando o aprimoramento das técnicas de teste dos circuitos.

II. Alguns conceitos de "Concepção visando a Testabilidade".

Sob o nome de "Concepção visando a Testabilidade", CVT (do inglês "Design for Testability"), várias técnicas são reunidas, cujos objetivos e campos de aplicação diferenciam-se totalmente.

Uma classificação inicial pode ser feita de acordo com o tipo de capacidade de teste que o método CVT fornece ao circuito: autoteste em funcionamento, autoteste fora de funcionamento,ou técnicas não relacionadas com autoteste.

Os testes em funcionamento são concorrentes com a operação normal do circuito (do inglês, "on-line") e os testes fora de funcionamento são aqueles cuja execução necessita suspender a operação normal do circuito (do inglês, "off-line").

Os métodos usados para atingir a capacidade de teste em funcionamento são completamente diferentes daqueles usados para o teste fora de funcionamento. Os primeiros baseiam-se em informação redundante (codificação), redundância do circuito (duplicação) ou redundância de tempo (auto-

comparação), enquanto que os últimos baseiam-se na reconfiguração funcional da arquitetura do chip.

Fora de funcionamento, o circuito é: facilmente testável, se o seu procedimento de teste deve ser executado por um monitor externo; e autotestável, se ele executa o seu próprio teste sem necessitar nenhuma supervisão externa.

Os circuitos facilmente testáveis contêm uma lógica extra usada para gerar padrões de teste e executar os testes facilmente. Entretanto, necessitam de dispositivo externo para gerar os estímulos de entrada e verificar a correção da resposta. Outros, mais complexos, são capazes de gerar, internamente, parte ou todo o estímulo de teste, mas necessitam de avaliação externa da resposta; ou, se podem realizar auto-avaliação das saídas de teste, necessitam da aplicação externa de padrões de teste.

Os circuitos autotestáveis possuem internamente toda a lógica necessária à geração dos estímulos de entrada, a coletar respostas e avaliá-las. O testador externo só é usado para iniciar o procedimento de teste, esperar por sua realização e então controlar o resultado (dado, em geral, por uma indicação externa do tipo "prosseguir" ou não).

As metodologias de concepção associadas a testes executados em períodos intercalados à operação do circuito, visam a detecção de falhas de fabricação ou daquelas surgidas como consequência do esgotamento da vida útil do componente ou da má-utilização de equipamentos ou componentes. Mas em sistemas críticos, onde a ocorrência de uma falha pode resultar em consequências graves, não podemos admitir que resultados errados sejam gerados e reutilizados dentro dos circuitos. Se este é o caso, é imprescindível o emprego de técnicas de autoteste em funcionamento, as quais permitem a detecção e sinalização de falhas, antes que essas resultem em erros perigosos. Exemplos desses tipos de sistemas podem ser: complexos de transporte de contrôle eletrônico (trens, metrôs, etc...), aviação, aplicações espaciais e militares, etc... Na maior parte delas, erros podem trazer como consequência a perda de vidas humanas.

Nas seções que se seguem, abordaremos conceitos relacionados aos sistemas autotestáveis que detectam falhas durante o funcionamento, sinalizando-as. O objetivo básico é a "tomada de consciência da falha". A reconfiguração do sistema, a entrada em operação de uma unidade suplementar, ou outras providências, são deixadas à área de "Tolerância a Falhas".

III. Hipóteses de falhas

Durante um longo tempo, o modelo de falhas mais frequentemente considerado em técnicas de teste era baseado na suposição de falhas lógicas representáveis por valores fixos nos níveis lógicos 0 ou 1 (em inglês, "stuck-at-0" "stuck-at-1"), associados ao diagrama lógico do circuito a ser testado. Mas existem alguns tipos de falhas não modeláveis por "colagens" ("stuck-at"). Na figura 1 é mostrado um exemplo, através do qual Galiay et al. [Gal 79] mostrou um caso não modelável por colagem - a falha implica na modificação da função executada pela porta lógica. O curto circuito 1 e o circuito aberto 2 não podem ser representados sobre o circuito lógico correspondente.

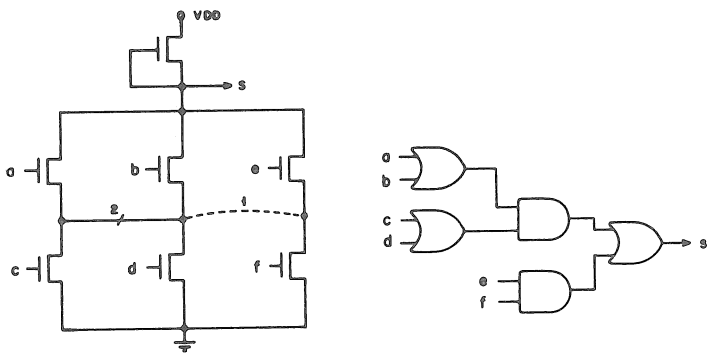


Figura 1: Falhas não representáveis por colagens.

Na prática, aparentemente a maior parte das falhas físicas são curtos circuitos e circuitos abertos (cortes nas linhas). Para esses, a representação do circuito por diagramas lógicos não é mais adequada. O estudo dos tipos de falhas deve ser embasado na topologia real do circuito, i.e., no modelo real do circuito físico. E os modelos de falhas devem representar falhas reais que ocorrem nos circuitos integrados, o que é referido como "hipóteses de falhas físicas" (ou "reais", ou "de baixo nível").

Para o projeto do circuito propriamente dito, outras considerações adicionais devem ser feitas, como: somente uma falha física ocorre em cada momento; há um certo espaçamento temporal entre a ocorrência de duas falhas físicas, etc...

IV. O sistema autotestável-teoria clássica

Conforme abordamos anteriormente, a metodologia de concepção autotestável emprega redundância de informação e material. Faz-se necessário

codificar informações internas para que "irregularidades" sejam detectadas, e incluir circuitos que atuarão nesta detecção. Assim a função básica inicial do circuito é executada por um circuito funcional construído sob determinadas regras e fornece um conjunto de saídas codificadas - estas serão decodificadas ou usadas diretamente por outros circuitos. O conjunto codificado é analisado por um controlador, o qual produz uma sinalização de erro, se for o caso. O sistema assim composto é representado pelo esquema da figura 2. Para que o mesmo tenha capacidade de detecção de 100% das falhas perigosas (que causam modificações da função executada), é necessário que este tipo de falhas cause modificação no código de saída do circuito funcional e/ou que o controlador garanta a indicação de erro, mesmo em presença de falha.

Com esse propósito são definidos os sistemas totalmente autotestáveis [AND 71]. São consideradas redes lógicas combinacionais G de múltiplas entradas e múltiplas saídas. O espaço de entrada X é composto de 2^r vetores binários de comprimento r, para r linhas de entrada. O espaço de saída Y é composto de 2^q vetores binários de comprimento q, para q linhas de saída. Não ocorrendo falha, a rede G recebe somente um subconjunto de X denominado espaço de código de entrada A e produz um subconjunto de Y denominado espaço de código de saída B. Elementos pertencentes ao espaço de código são referidos como palavras codificadas. Palavras não codificadas podem ser geradas sob a ocorrência de uma falha.

A saída de G sob uma entrada x é representada por: G(x,f), para uma falha f presente na rede G e G(x,∅), se não há falha na rede G.

Definição 1: G é segura a falhas com relação a F se para todas falhas em F e todas entradas de código, a saída ou é correta ou é uma palavra não codificada; i.e., para todas f ε F e para todas a ε A,

$$G(a,f) = G(a,\emptyset) \text{ ou } G(a,f) \notin B.$$

Definição 2: G é autotestável com relação a F se para cada falha em F há ao menos uma entrada de código que produz uma saída não codificada; i.e., para todos fεF, há uma entrada aεA, tal que G(a,f)∉B.

Definição 3: G é totalmente autotestável (TAT) com relação a F se ela é segura a falhas e autotestável com relação a F.

Portanto, de acordo com as definições acima, uma saída errada provocada por uma falha em um circuito seguro a falhas é sempre um vetor não codificado. E um circuito autotestável é diagnosticável com apenas suas entradas de código, e ao menos uma entrada de código resulta em uma saída não codificada para qualquer falha. Assim, em circuitos que são simultaneamente autotestáveis e seguros a falhas (circuitos TAT), todas as falhas

causam saídas errôneas detectáveis durante sua operação normal.

Definição 4:Uma rede G é de código disjunto se ela sempre mapeia entradas codificadas em saídas codificadas, e entradas não codificadas em saídas não codificadas, i.e., se:

$\forall a \in A,\ G(a,\emptyset) \in B$

$\forall a \notin A,\ G(a,\emptyset) \notin B.$

Definição 5: Uma rede é um controlador TAT se ela é TAT e de código disjunto.

O objetivo dos sistemas construídos com as características definidas acima é de assegurar que a primeira saída incorreta obtida, não é uma palavra do espaço de código de saída B. Esta propriedade é referida como objetivo TAT.

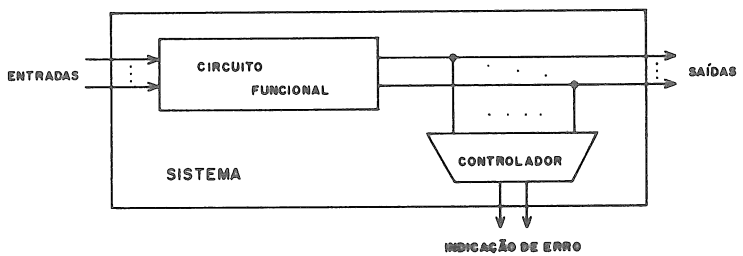


Figura 2: Estrutura de um Sistema autotestável.

V. O caso real (teoria moderna)

Quando são consideradas hipóteses de falhas físicas, sabe-se que podem ocorrer determinadas falhas que não causam erros na saída, como certos tipos de curtos circuitos entre linhas de mesmo tipo, cortes em uma linha redundante, etc... Entretanto estes casos não são previstos pela teoria clássica (redundância com relação a determinadas falhas).

Da mesma maneira, a teoria clássica, calcada sobre portas lógicas como elemento base, podia considerar a utilização de circuitos integrados comerciais como elementos de implementação, cujas entradas e/ou saídas estariam com valores fixos em 0 ou 1, dependendo da falha ocorrida. Atualmente sabe-se que o emprego de portas complexas internas a um mesmo chip cuja estrutura interna não se conhece,pode resultar em modificações de função não representáveis através dos parâmetros e variáveis disponíveis como entradas e saídas deste circuito.

A partir dessas considerações foram definidos os circuitos funcionais "fortemente seguros a falhas" (do inglês, "strongly fault secure",

[SMI 78]), e os controladores "de código fortemente disjunto" (do inglês, "strongly code disjoint", [NIC 84b]). Esses circuitos são definidos a partir de hipóteses de falhas físicas, e são concebidos sob determinadas restrições (regras). A principal diferença, com relação à teoria clássica, refere-se a possibilidade de ocorrerem falhas com relação as quais o circuito é redundante, como definido a seguir.

Definição 6: Uma rede que realiza a função $G(a,\emptyset)$ é redundante com relação a falha f se $G(a,f) = G(a,\emptyset)$, $\forall a \in A$.

Pode-se observar que uma rede redundante com relação a uma falha f não mantém a propriedade autotestável. Mas o maior problema é a possibilidade de ocorrência de uma segunda falha a qual, associada à primeira não detectada, pode provocar palavras de saída erradas. Lembramos que a segunda falha pode ser vista como não mais afetando a rede G, mas uma rede G modificada pela primeira falha. Ou podemos considerar G como sendo afetada por uma falha dupla. Assim, o conceito de circuitos TAT não é mais aplicável: ele deve ser substituído por outro que, admitindo a ocorrência de falhas redundantes, ainda permita ao sistema atingir o objetivo TAT.

Definição 7: Para uma sequência de falhas $<f1, f2, \ldots, fn>$, seja k o menor inteiro para o qual existe $a \in A$ tal que

$$G(a, \bigcup_{j=1}^{k} fj) \neq G(a,\emptyset).$$

Se não existe tal k, seja k=n. Então G é fortemente seguro a falhas (FSF) com relação à sequência de falhas se, para todas as palavras de código $a \in A$,

$$\text{ou } G(a, \bigcup_{j=1}^{k} fj) = G(a,\emptyset) \text{ ou } G(a, \bigcup_{j=1}^{k} fj) \notin B.$$

No caso dos controladores, para que o circuito seja redundante com relação a determinada falha, ele deve manter sua propriedade de "código disjunto". No caso do controlador denominamos B o seu espaço de código de entrada (é o espaço de código de saída do circuito funcional) e C, o seu espaço de código de saída.

Definição 8: Um controlador G é redundante com relação a uma falha f e com relação a um espaço de código de entrada B e com relação a um espaço de código de saída C se:

$\forall b \in B$, $G(b,f) \in C$ e

$\forall b \notin B$, $G(b,f) \notin C$.

Definição 9: Antes que ocorram falhas, o circuito G é de código disjunto. Para uma sequência de falhas $<f1, f2, \ldots, fn>$, seja k o menor inteiro para o qual existe um vetor $b \in B$ tal que:

$$G(b, \bigcup_{j=1}^{k} fj) \notin C.$$

Se não existe tal k, façamos k=n. Assim, o circuito G é "de código fortemente disjunto"(CFD) para a sequência <f1,f2,..., fn> se:

$$\forall b \in B, \forall m \in \{1,2,\ldots,k-1\}, G(b, \bigcup_{j=1}^{m} fj) \notin C.$$

Um circuito funcional FSF é seguro a falhas mesmo em presença de falhas com relação às quais ele é redundante. Um controlador CFD mantém a propriedade de código disjunto também em presença de falhas com relação as quais ele é redundante. Sob a suposição de que não ocorrem falhas ao mesmo tempo no circuito funcional e no controlador, isto é,que após a ocorrência de uma falha no circuito funcional ou no controlador, decorre um espaço de tempo suficiente para a aplicação das entradas de código do respectivo bloco, e que não ocorre nenhuma outra falha, pode-se verificar que o objetivo TAT é atingido pelo sistema assim composto.

VI. Aplicação e conclusões

Os circuitos funcionais FSF são construídos sob determinadas regras que lhes conferem propriedades de detecção de erros simples, unidirecionais ou múltiplos [NIC 84a]. A codificação interna deve ser também coerente com a capacidade de detecção desejada. As estruturas internas de blocos diversos como memória, PLA, unidade logica-aritmética, decodificadores dão margem à ocorrência de diferentes tipos de erros. Assim, se o tipo de erros produzidos é simples, usa-se um código de paridade; para erros unidirecionais produzidos, usa-se códigos m-entre-n, Berger ou Berger modificado; onde erros múltiplos podem ser gerados, um código double-rail garante a detecção.

Na saída dessas unidades, o controle da codificação correta é executada por controladores CFD. Para esses controladores, não existem regras específicas de construção - eles devem ser analisados individualmente em sua topologia interna. O emprego de células de base em sua construção é uma possibilidade que auxilia enormemente a agilização dessa atividade [JAN 84].

A redundância introduzida pela nova concepção dos circuitos ou pela inclusão de outros é variável conforme o tipo de unidades funcionais especificadas. Em média, não ultrapassa 60% da superfície inicial. Um estudo executado sobre o MC 68000 [NIC 84a], um circuito processador geral, resultou em um aumento de superfície em 48%, assegurando a detecção de 100% das falhas simples - inclui corte de linhas, grade flutuante, falha em contato ou pré-contato, e curto-circuito entre duas linhas de alumínio ou duas de difusão. Estes são defeitos reais passíveis de ocorrer em circuitos cuja tecnologia de construção é NMOS.

De acordo com os métodos clássicos, esta proteção só seria atingida através da duplicação integral do circuito, o que implicaria 100% de aumento da superfície e um controlador de saída para executar a comparação.

Referências

[AND 71] ANDERSON, D.A. Design of self-checking digital networks using coding techniques. Urbana, CSL/Univ. Illinois, Set.1971. (Report 527). 133p.

[GAL 79] GALIAY, J.; CROUZET, Y.; VERGNIAULT, M. Physical versus logical fault models in MOS LSI circuits. Impact on their testability. FTCS-9, Madison, 20-22/junho/79. New York, IEEE, 1979. p.189-192.

[JAN 84] JANSCH, I. & COURTOIS,B. Design of SCD checkers based on analytical fault hypothèses. ESSCIRC'84, Edinburgh, 19-21 /setembro/84. Edinburgh, CEP Consultants Ltd., 1984. p. 109-12.

[NIC 84a] NICOLAIDIS, M. Conception de circuits integrés autotestables pour des hypothèses de pannes analytiques. Grenoble, INPG, Jan. 1984 (Tese).

[NIC 84b] NICOLAIDIS, M.; JANSCH, I.; COURTOIS, B., Strongly code disjoint checkers. FTCS-14, Kissimmee, 20-22/junho/84. New York, IEEE, 1984. p. 16-21.

[SMI 78] SMITH, J.E.; METZE, G. Strongly fault secure logic networks. IEEE Transactions on Computers. New York, IEEE, 27(6): 491-9, Junho/1984.

UM EQUIPAMENTO AUTOMATICO DE TESTES PARA MANUTENCAO E QUALIFICACAO

MARCELO D. P. SILVEIRA "

RESUMO

Este trabalho apresenta um equipamento automatico de teste destinado a equipamentos digitais. A sua caracteristica predominante e' a utilizacao do processador do proprio equipamento sob teste como escravo para aumentar a sua testabilidade.

ABSTRACT

This paper presents an automatic test equipment (ATE) for digital equipment. Its main feature is to use the processor of the equipment under test as a slave to improve its testability.

" Engenheiro Eletronico (UFRGS, 1984); Aeromot/Divisao Aeroeletronica, Caixa Postal 8031, Porto Alegre RS

INTRODUCAO

O equipamento automatico de teste em desenvolvimento na Aeroeletronica partiu da necessidade de se detectarem possiveis falhas em equipamentos avionicos. Estes sao testados ao final de uma linha de montagem, quando sujeitos a testes de qualificacao (vibracao, choques, variacao de temperatura e umidade, etc.) e quando forem ser instalados na aeronave. O equipamento e' ligado ao sistema de teste ao mesmo tempo em que e' submetido a uma destas situacoes e, no caso de existir alguma falha, o sistema de teste avisa em alguns minutos o tipo de falha e a placa que deve ser substituida ou reparada.

Este sistema foi concebido para testar unidades específicas que estao em fase final de desenvolvimento mas pode ser facilmente adaptado 'a maioria dos sistemas digitais com microprocessador. As unidades referidas acima fazem parte de um sistema maior e possuem paineis e conectores para interliga'-las ao restante do sistema. O equipamento que deve ser testado e' ligado ao sistema de teste apenas por estes conectores. Nao e' necessario criar nenhum sinal exclusivamente para a interface com o sistema de teste. Este utiliza apenas os sinais que sao usados pelo equipamento em funcionamento normal.

A interface do sistema de teste (ST) com o equipamento sob teste (EST) nao e' digital. Os sinais do EST, ao serem lidos pelo ST, sao na realidade comparados com valores de tensao gerados por um conversor digital/analogico. Estas tensoes sao selecionadas conforme o tipo de saida que esta' sendo testada. Uma saida digital, por exemplo, e' entendida como em nivel '0' ou '1' se estiver dentro da faixa de tensao correspondente testando sempre os valores mais criticos dentro desta faixa. Esta caracteristica, alem de oferecer maior confiabilidade aos testes da parte digital, permite testes em partes analogicas de um equipamento.

TESTES REALIZADOS

A seguir sao mencionados alguns tipos de testes que podem ser realizados atraves deste equipamento:

- processador
- circuito de relogio e osciladores
- PROM, RAM e portas de E/S
- timers
- drivers, buffers e outros circuitos MSI e SSI
- conversores D/A e A/D
- fluxo de dados em um barramento padrao
- circuito impresso
- fiacao ate' os conectores
- paineis (chaves, indicadores luminosos)

DESCRICAO DO EQUIPAMENTO

Um blocodiagrama do equipamento e' apresentado na figura 1. Ele possui um painel onde e' exibido o resultado dos testes atraves de displays e indicadores luminosos. Ha' uma interface RS-232C para a exibicao de resultados mais detalhados em uma impressora. Esta' prevista a ligacao de um terminal com video/teclado. Ha' tambem uma linha serial para comuni-

cacao com o equipamento sob teste (EST). No caso de teste de equipamentos que nao possuam uma porta serial, esta comunicacao pode se dar de outra forma.

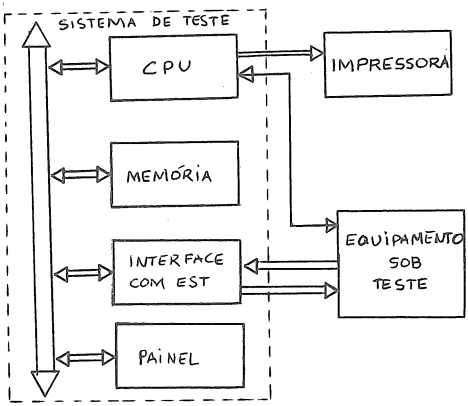


fig. 1 - Blocodiagrama do ST

A interface com o equipamento sob teste (EST) e' especifica para cada tipo de equipamento que vai ser testado. Nela existem circuitos que simulam a carga real para as quais as saidas do EST foram projetadas e que dao o nivel de tensao apropriado para cada tipo de sinal de entrada e saida.

Com esta arquitetura, o sistema de teste (ST) pode conversar com a unidade sob teste enviando ordens a serem executadas por esta e recebendo respostas 'as ordens enviadas previamente. Os algoritmos de pesquisa de falhas tem como base esta interacao entre os processadores do sistema de teste e do equipamento que esta' sendo testado.

De um modo geral, os equipamentos de teste para sistemas digitais possuem algoritmos de pesquisa de falhas baseados em padroes de estimulo/resposta. Estes padroes contem um conjunto de entradas e as correspondentes saidas esperadas de um determinado circuito. Este pode ser uma porta logica, um circuito integrado, uma placa ou mesmo um equipamento completo.

O equipamento aqui descrito, alem de possuir acesso 'as entradas e saidas do equipamento sob teste, comunica-se com o processador deste. Assim, a testabilidade de um equipamento aumenta muito porque os padroes de estimulo/resposta nao se limitam 'as entradas e saidas do sistema. Por exemplo, o estimulo pode ser aplicado pelo sistema de teste (ST) e a resposta, obtida pelo processador do EST, e' enviada ao ST para analise; o ST pode ordenar o processador do EST aplicar um estimulo e a resposta ser analisada pelo ST; o ST pode ordenar ao EST a execucao de um teste e esperar a resposta deste dizendo se o teste foi executado com sucesso ou nao.

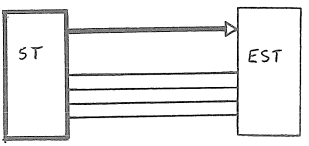
TOPICOS SOBRE O ALGORITMO DE PESQUISA DE FALHAS

Conforme o que foi apresentado ate' entao, torna-se evidente que o equipamento sob teste precisa ter algum software especifico para a realizacao dos testes. Nos equipamentos da Aeroeletronica, optamos por ter este software na propria EPROM que contem o software funcional do equipamento. Este, quando energizado, precisa testar algumas condicoes para saber se esta' em operacao normal ou sob teste. Uma vez sob teste, ele passa a operar como escravo do sistema de teste, executando os comandos recebidos pela linha de comunicacao entre os dois.

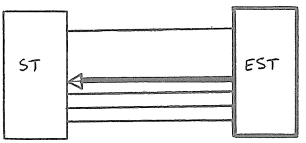
Foi, entao, definido um conjunto de primitivas basicas de comunicacao entre os processadores e de execucao de testes. Sao primitivas simples

e modulares de forma que, a partir delas, se possa construir um algoritmo complexo que teste o equipamento desejado com um mínimo de software e hardware específico a cada equipamento testado. Vamos exemplificar para se ter uma ideia mais clara.

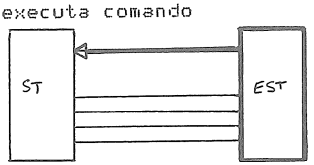
Primitiva que testa uma porta de saída



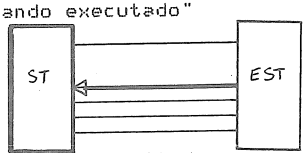
3a - ST envia mensagem



3b - EST executa comando



3c - EST envia mensagem de "comando executado"



3d - ST analisa resultado

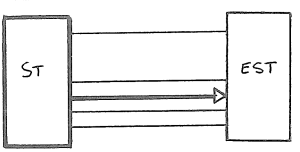
O sistema de teste (ST) envia uma mensagem simples ao equipamento sob teste (EST) dizendo que este deve escrever um valor binário em uma determinada porta de saída (fig. 3a). O EST executa o comando (fig. 3b) e devolve uma resposta ao ST dizendo se o comando foi executado (fig. 3c). O ST, então, le o valor contido na saída do EST e compara com o valor previamente enviado (fig. 3d).

Com esta sequencia simples de operacoes são testados decodificadores, portas, drivers e outros circuitos MSI e SSI envolvidos na geracao de um determinado sinal de saída desde o microprocessador até o conector.

Com uma composicao adequada de diversos comandos deste tipo podem-se testar todas as combinacoes de valores nas saídas do EST e verificar se algum sinal varia quando se modifica o estado de todos os outros sinais separadamente. Desta forma pode-se, por exemplo, detectar e isolar falhas como curto-circuitos entre estes sinais.

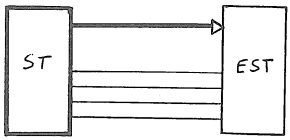
fig.3 - Sequencia de execucao de uma primitiva de teste de saída

Primitiva que testa uma porta de entrada

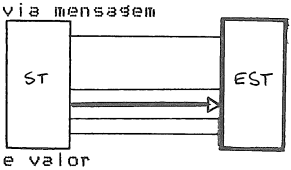


4a - ST escreve valor

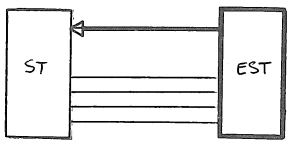
O ST escreve um valor em uma entrada do EST (fig. 4a) e envia uma mensagem dizendo ao EST para ler esta entrada (fig. 4b) O EST le o conteudo da porta de entrada especificada pelo ST (fig. 4c) e envia o valor lido



4b - ST envia mensagem



4c - EST le valor



4d - EST envia resposta com o valor lido

para o ST (fig. 4d) que o analisa.

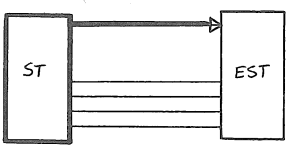
Da mesma forma que no caso anterior, combinando comandos deste tipo podem-se testar todas as possibilidades de sinais de entrada e saida. Assim, testam-se os circuitos ja mencionados anteriormente e detectam-se eventuais curto-circuitos. Pode-se ainda verificar a influencia de pulsos ou trem de pulsos nas entradas do EST.

fig. 4 - Exemplo de uma primitiva de teste de entrada

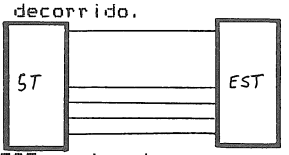
Teste de conversores D/A e A/D

Estes testes sao realizados de forma analoga aos anteriores. Consideremos que o EST tenha em alguma saida a tensao gerada por um conversor D/A. O ST envia uma mensagem dizendo ao EST para escrever no conversor um determinado valor. O ST, apos ter recebido uma mensagem do EST confirmando a escrita daquele valor, compara aquela tensao analogica com 2 valores de tensao gerados pelo seu conversor D/A. Estes valores devem ser um pouco acima e um pouco abaixo do esperado com uma diferenca igual ao erro maximo permitido. Assim, pode-se testar toda a faixa de valores de um conversor e, com estes dados, verificar suas caracteristicas como linearidade, monotonicidade, tensao de offset do conversor, etc.

Teste do circuito de relogio



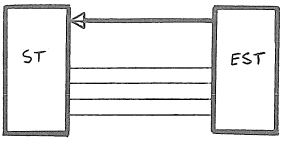
5a - O ST manda EST marcar o tempo decorrido.



5b - ST e EST contam tempo.

Equipamentos submetidos a condicoes adversas (choques, vibracoes, etc.) podem ter seus circuitos de relogios ou osciladores fora das especificacoes.

O seguinte teste pode verificar se eles estao funcionando como desejado: o ST manda uma mensagem ao EST (fig. 5a) dizendo para este esperar um determinado intervalo de tempo (fig. 5b) ao fim do qual o EST deve sinalizar ao ST (fig.



5c - EST avisa fim do intervalo de tempo

5c). Neste momento o ST compara o tempo decorrido de acordo com o seu relogio com o intervalo de tempo marcado pelo EST.

fig. 5 - Exemplo de um teste de relogio

Um sistema de teste necessita ser configurado para cada tipo de equipamento que vai ser testado. Uma das configuracoes que se fazem necessarias e' o software de testes especificos a cada um. Primitivas, como as apresentadas aqui, tornam esta tarefa muito simples pois se desenvolve toda uma bateria de testes a partir de modulos simples e bastante genericos.

CONCLUSAO

O sistema de teste apresentado foi projetado para testes de hardware de equipamentos digitais com microprocessador. Como um teste de hardware nao envolve a parte funcional do equipamento, optou-se por um sistema de teste que usa os proprios recursos computacionais do equipamento sob teste para aumentar a sua testabilidade.

Obviamente o sistema de teste necessita ser configurado para cada tipo de equipamento a ser testado. No nosso caso, estas configuracoes se limitam a uma placa de interface com o equipamento a ser testado e a parte de software especifico para este equipamento. No caso de equipamentos digitais sem microprocessador, este sistema tambem pode ser utilizado da forma mais convencional com algoritmos de pesquisa de falhas com estimulo/resposta externas.

Atualmente, este equipamento esta' em fase de desenvolvimento. Algumas partes ja' estao prontas e com algum software instalado permitindo a realizacao de testes em prototipos das unidades que precisarao ser testadas por este equipamento.

Quando estiver concluido, este sistema de teste sera' empregado em diversos niveis de manutencao e qualificacao de equipamentos para aviacao que requerem alta confiabilidade.

BIBLIOGRAFIA

Hewlett-Packard Journal, Oct 1984.

Stout,D."Microprocessor Applications Handbook",1982.

H. P. NEVES *

SUMÁRIO

O trabalho a seguir descreve um analisador espectral baseado em microcomputador, com aplicação dirigida mais especificamente à análise de padrões vocais. Apesar dessa especificidade, não se procurou desenvolver os algoritmos propriamente ditos no que se refere à identificação dos padrões da voz humana, mas sim fornecer uma ferramenta a essa análise. Cabe ressaltar que o analisador poderia encontrar aplicações diversas, tais como análise espectral em física nuclear, reconhecimento de imagens, análise de padrões em radar e sonar, etc.

ABSTRACT

The paper describes a microcomputer-based spectral analyzer, designed specifically for vocal pattern analysis. In spite of its specificity, there had been no intention to deal with the pattern analysis algorithms themselves, but to make a tool for such an analysis. It should be stressed that our analyzer can find several different applications , like spectral analysis applied to nuclear physics, image recognition, pattern analysis applied to radar and sonar, etc.

* Engenheiro eletricista (Universidade Federal de Minas Gerais, 1985); microeletrônica e interfaces especiais para microcomputadores; trabalho desenvolvido no Depto. de Engenharia Eletrônica da UFMG em 1984; Rua Síria, 116 - Floresta - 30000 Belo Horizonte, MG

INTRODUÇÃO

No desenvolvimento desse trabalho, foi utilizado um microprocessador Z80, o qual foi escolhido por sua grande versatilidade no que se refere ao seu conjunto de instruções, e pela facilidade de encontrá-lo no mercado brasileiro. A interface para a entrada analógica de sinal é realizada por um conversor analógico digital fabricado pela Analog Devices, o AD571, que é um conversor rápido de 10 bits, capaz de efetuar uma conversão em apenas 25 microssegundos. O microprocessador executa uma rotina de transformada rápida de Fourier para a conversão de domínio tempo-frequência, além de gerenciar a aquisição de dados. Tal rotina de cálculo foi escolhida devido à sua grande velocidade, o que permite sua utilização em tempo real.

HARDWARE

Utilizou-se para a conversão analógico-digital um conversor AD571, que é um conversor rápido de 10 bits (tempo de conversão = 25 μs), tendo-se em vista que, para aplicações diferentes da original (processamento de voz) velocidades de conversão maiores podem ser necessárias. Pode-se demonstrar que

$$F = \frac{1}{(t \cdot N)}$$

onde F é a distância (em Hertz) entre as raias espectrais desejadas, t o intervalo de amostragem e N o número de pontos a serem processados. No nosso caso, serão processados 1024 pontos por amostragem. Se tivermos t = 1 ms, teremos uma separação entre raias de cerca de 1 Hz, o que nos dá uma excelente resolução espectral. Utilizou-se como circuito de entrada um microfone de eletreto com dispositivo ativo incorporado e dois amplificadores operacionais. O primeiro deles (AO1) encarrega-se de amplificar o sinal recebido do microfone, ajustando-o a um nível adequado, enquanto o segundo (AO2) atua como buffer a fim de adaptar a impedância do circuito com a entrada do AD571, cuja impedância é relativamente baixa (aproximadamente 5000 Ω). Os CI's 2 e 3 (74LS373) são latches octais que travam o resultado da leitura do conversor A/D. Isso se faz necessário pois o conversor trabalha com dados de 10 bits enquanto que o barramento de dados do Z80 é de 8 bits. Assim, toda vez que uma conversão se completar, os latches são trancados através do sinal proveniente do pino 17 do conversor (aviso de fim de conversão, "READY"), já que este encontra-se ligado ao pino ENABLE G de cada um dos latches. Observe-se que tal pino encontra-se também ligado ao pino INT do Z80. Isso porque toda vez que o computador estiver pronto para carregar dados novos na memória, ele entra em Halt, ou seja, pára de processar e fica aguardando uma interrupção. Ao entrar em Halt, a linha HALT do Z80 é ativada e, como ela está conectada à linha 11 do conversor (que comanda o início de uma nova conversão, "CLEAR/CONVERT"), ela comanda-o a executar uma

conversão. Terminada esta, a linha 17 do conversor é ativada para sinalizar "fim de conversão". Estando ela ligada à linha INT do computador, aciona então uma interrupção, fazendo com que o dado recém entrado seja armazenado. Foi usado para armazenamento dos dados um banco de memória dinâmica (8 CI's 4116) de 8 Kbytes, já que são processados 1024 pontos, os quais, em notação de ponto flutuante de 4 bytes, ocupam 4096 bytes, devendo-se ainda considerar que tais pontos são tratados como números complexos, necessitando pois de uma área igual à anterior para armazenar a parte imaginária. Foi ainda incorporada RAM estática de 1 Kbyte (2 CI's 2114) destinado às variáveis de cálculo, à pilha do Z80 e à pilha operacional destinada às operações com ponto flutuante. Além disso, foi usada uma EPROM 2732 (8 K x 8), cuja capacidade de armazenamento é mais que suficiente para abrigar o programa total (foi aqui levado em consideração o fato de que o programa em questão é normalmente utilizado como suporte de um programa global, devendo-se portanto destinar-lhe espaço). Utilizou-se ainda um multiplexador (MUX) para o endereçamento e refrescamento das memórias dinâmicas. A figura 1 apresenta o hardware do sistema.

SOFTWARE

a) Descrição da transformada rápida de Fourier

A transformada rápida de Fourier (em inglês "Fast Fourier Transform"- FFT) é um poderosíssimo recurso surgido em 1965 para se calcular a transformada discreta de Fourier com uma velocidade incrivelmente maior que no processo convencional, facilitando assim sua aplicação em tempo real. Para que se tenha uma base de comparação, o processo normal de cálculo de uma transformada de Fourier exigiria, para uma função amostrada em 1024 pontos (como no nosso caso) cerca de quatro milhões de multiplicações, enquanto que no cálculo através da FFT esse número cai para menos de 20000 multiplicações. A razão entre os tempos de processamento dos dois métodos pode ser demonstrada como sendo

$$R = 2.N / \gamma$$

onde N é o número de pontos e γ é o logaritmo na base 2 de N. Para N=1024, γ=10 e consequentemente R=204,8; portanto o programa usando FFT é 200 vezes mais rápido que o convencional. A figura 2 ilustra através de um gráfico a diferença entre os tempos.

Mostraremos a seguir os fundamentos nos quais se baseia a transformada rápida de Fourier. Seja a transformada convencional contínua de Fourier de uma função contínua s(t):

$$S(f) = \int_{-\infty}^{\infty} s(t)\, e^{-j2\pi ft}\, dt$$

onde S(f) é a transformada s(t). Fazendo-se a convolução da função s(t) com a função

com a função delta de Dirac, teremos s(t) discretizada no tempo. Após truncarmos tal função amostrada num intervalo T, obtemos sua transformada e fazemos a convolução desta com a função delta de Dirac no domínio da frequência, obtendo assim a definição da transformada discreta de Fourier, ou seja,

$$G(\frac{n}{NT}) = \sum_{k=0}^{N-1} g(kT)e^{-j2\pi nk/N}$$

onde k=0,1,2,3,...,N-1. Escrevendo-se k em vez de kT e n em vez de n/NT, obtemos a representação simplificada abaixo:

$$X(n) = \sum_{k=0}^{N-1} x_0(k)e^{-j2\pi nk/N}$$

onde n=0,1,2,3,...,N-1. Podemos notar que a equação acima descreve o cálculo de N equações. Se N=4 e se nós escrevermos W=exp(-jπ/N), podemos reescrevê-la como:

$$\begin{aligned}
X(0)&=x_0(0)W^0+x_0(1)W^0+x_0(2)W^0+x_0(3)W^0\\
X(1)&=x_0(0)W^0+x_0(1)W^1+x_0(2)W^2+x_0(3)W^3\\
X(2)&=x_0(0)W^0+x_0(1)W^2+x_0(2)W^4+x_0(3)W^6\\
X(3)&=x_0(0)W^0+x_0(1)W^3+x_0(2)W^6+x_0(3)W^9
\end{aligned}$$

que é o mesmo que

$$\begin{bmatrix} X(0)\\ X(1)\\ X(2)\\ X(3)\end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0\\ W^0 & W^1 & W^2 & W^3\\ W^0 & W^2 & W^4 & W^6\\ W^0 & W^3 & W^6 & W^9\end{bmatrix} \cdot \begin{bmatrix} x_0(0)\\ x_0(1)\\ x_0(2)\\ x_0(3)\end{bmatrix} \qquad (1)$$

É importante observar aqui que, por razões que serão vistas posteriormente, é conveniente escolher o número de pontos de amostragem sempre como uma potência de 2. Baseados na propriedade de que $W^{nk}=W^{(nk\ MOD\ N)}$ onde nk MOD N é o resto da divisão de nk por N , podemos reescrever a equação (1) como:

$$\begin{bmatrix} X(0)\\ X(1)\\ X(2)\\ X(3)\end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1\\ 1 & W^1 & W^2 & W^3\\ 1 & W^2 & W^4 & W^6\\ 1 & W^3 & W^6 & W^9\end{bmatrix} \cdot \begin{bmatrix} x_0(0)\\ x_0(1)\\ x_0(2)\\ x_0(3)\end{bmatrix}$$

podemos demonstrar essa propriedade para, por exemplo, nk=6:

$$W^6 = e^{(-j2\pi/4).6} = e^{-j3\pi} = e^{-j\pi} = e^{(-j2\pi/4).2} = W^2$$

fatorando-se a matriz temos

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \cdot \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \quad (2)$$

a forma de se fatorar a matriz é que caracteriza a rotina de FFT. Multiplicando-se as duas últimas matrizes, temos:

$$\begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \cdot \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix}$$

podemos notar que, no cálculo de $x_1(0)$ e $x_1(2)$, temos

$$x_1(0) = x_0(0) + W^0 x_0(2)$$
$$x_1(2) = x_0(0) + W^2 x_0(2) = x_0(0) - W^0 x_0(2)$$

o que implica em 2 somas e 1 multiplicação; o mesmo se verifica para $x_1(1)$ e $x_1(3)$. Portanto economizamos 2 multiplicações redundantes. Continuando o cálculo, multiplicamos as duas matrizes restantes:

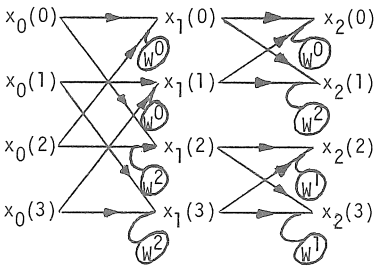
$$\begin{bmatrix} x_2(0) \\ x_2(1) \\ x_2(2) \\ x_2(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \cdot \begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix}$$

Observe-se que a fatoração obtida em (2) levou a uma inversão entre as linhas 2 e 3; tal fenômeno faz parte da fatoração e devemos ter o cuidado de, após terminar os cálculos obtendo a equação acima, desinverter as linhas adequadamente. Para isso, é suficiente que se escreva a ordem de cada linha em binário e que se reescreva cada ordem de trás para frente, conforme abaixo:

0 → 00 → inversão → 00 (0)
2 → 10 → inversão → 01 (1)
1 → 01 → inversão → 10 (2)
3 → 11 → inversão → 11 (3)

Vejamos agora como é feita a fatoração da matriz principal. Podemos representar cada etapa do processo graficamente, sendo que os resultados intermediários, a partir da matriz das variáveis independentes, são escritos verticalmente e as operações representa-

das por setas. Dessa forma, o cálculo da transformada acima exemplificado (N=4) é representado da seguinte forma:



Observe-se que a primeira coluna representa a matriz das variáveis independentes, a segunda representa o primeiro resultado parcial, correspondente ao produto das duas últimas matrizes da equação (2) e a terceira coluna representa o produto final. As linhas que convergem para $x_1(0)$, por exemplo, significam que ele é obtido a partir da soma de $x_0(0)$ e $x_0(2)$, este último multiplicado por W^0 (note que a seta proveniente de $x_0(2)$ tem W^0escrito debaixo dela). Podemos também notar que existem pares de valores que são calculados a partir dos mesmos valores anteriores, como no caso de $x_1(0)$ e $x_1(2)$, que são calculados a partir de $x_0(0)$ e $x_0(2)$. Como estes últimos dois só são utilizados no cálculo de $x_1(0)$ e $x_1(2)$, podemos logo deduzir que cada coluna vertical de dados acima pode ser armazenada no mesmo local aonde estavam os dados que foram utilizados para calculá-la; dessa forma, se estamos trabalhando com uma amostragem de N pontos, necessitaremos somente de N posições de memória para armazenar os dados, desde a função original até a função transformada. A cada par de valores como descrito acima, chamamos nó dual. É importante dizer que os nós duais aparecem sempre com um espaçamento de $N/2^{\ell}$, onde N é o número de pontos amostrados e ℓ é a ordem do cálculo, isto é o número da coluna vertical com a qual estamos operando.

Outro passo importante é a determinação do expoente de W em cada operação. Para isso devemos seguir três regras:

1) Escrever o índice do elemento que desejamos calcular em binário usando γ bits;
2) Deslocar o número γ-ℓ bits para a direita zerando os espaços vagos;
3) Reverter a ordem dos bits.

Por exemplo, para γ=4, ℓ=3 e índice do elemento =2, temos:

0010 → deslocamento → 0001 → reversão → 1000

Na figura 3 temos um algoritmo simplificado da transformada rápida de Fourier.

b) Representação dos números em ponto flutuante

Foi utilizada para finalidades de cálculo uma notação de ponto flutuante, onde os números são representados por números de 4 bytes, sendo que o primeiro deles é o expoente e os demais são a mantissa. Tal representação é mais que suficiente em termos de precisão para a maioria das aplicações conhecidas de FFT. Para representarmos um número em notação de ponto flutante da maneira que foi utilizada em nosso programa, basta que sigamos os seguintes procedimentos:

1) O zero é representado como 0000 0000 0000 0000 0000 0000 0000 0000.

2) Sem considerar o sinal, escreve-se o número em binário. Exemplos:

   -10 → 1010

   0,375 → 0,011

   4,25 → 100,01

   7 → 111

3) Se o número à esquerda da vírgula binária for zero e o primeiro dígito à sua direita for 1, o expoente será zero.

4) Se o número à esquerda da vírgula binária for zero e o primeiro dígito à sua direita for 0, o expoente será igual ao negativo do número de zeros à direita da vírgula decimal até chegar no primeiro dígito igual a um.

5) Se o número à esquerda da vírgula binária for diferente de zero, o expoente será igual ao número de dígitos. Exemplos:

   -10 → expoente 4

   0,375 → expoente -1

   4,25 → expoente 5

   7 → expoente 3

6) Se o número a ser representado for negativo, deixamo-lo tal como ele está, se for positivo, trocamos por zero o primeiro bit da mantissa. Exemplos:

-10 → 0000 0100 1010 0000 0000 0000 0000 0000

0,375 → 1111 1111 0100 0000 0000 0000 0000 0000

4,25 → 0000 0101 1000 1000 0000 0000 0000 0000

7 → 0000 0011 0110 0000 0000 0000 0000 0000

c) Cálculo das funções trigonométricas

Para o cálculo das funções trigonométricas, foram utilizados polinômios de Chebyshev, que consistem em funções elementares que se aproximam com muita exatidão das funções desejadas. Eles podem ser definidos da seguinte forma:

$$T_{n+1}(z) = 2zT_n(z) - T_{n-1}(z)$$

onde n é a ordem do polinômio e z a variável independente. Para o cálculo de seno e cosseno, foi utilizado um valor de n=6. Podemos obter os coeficientes para as potências de z em qualquer compêndio de cálculo numérico avançado. Na figura 4 temos o gráfico dos

polinômios de Chebyshev até a quarta ordem. Cabe ressaltar que optou-se pelo uso desses polinômios pela velocidade de cálculo razoavelmente alta em relação aos processos convencionais, como por exemplo através de séries de Taylor, sendo que sua precisão é maior.

d) Programa principal

Na figura 5 temos o fluxograma geral do programa. Ele foi desenvolvido utilizando-se o programa Assembler EDTASM da Apparat Inc. Note-se que a parte correspondente ao FFT foi resumida em vista de já ter sido apresentada em maiores detalhes na figura 3. O programa inicia com a habilitação de interrupção para a aquisição de dados. Uma vez em Halt, o computador aguarda uma requisição de interrupção e fica em estado de espera. Cada vez que uma interrupção é solicitada (o que corresponde a um fim de conversão, como já vimos na seção de Hardware), o programa passa a ser executado no endereço 0038H (devido ao modo de interrupção utilizado), aonde temos a aquisição de dados propriamente dita. Cada vez que os dados são carregados entra-se novamente em Halt para aguardar outros dados. Isso não ocorre somente quando a área de dados já está cheia, o que faz com que a execução seja desviada para o corpo principal do programa, aonde os dados são convertidos para a notação de ponto flutuante, são processados pela rotina de FFT e são novamente reconvertidos para números inteiros, desta vez de 16 bits. Após essa reconversão, retorna-se ao início do programa para que novos dados sejam armazenados e convertidos. Como já foi dito no início das explicações, nosso programa funciona como subsídio a um programa global de reconhecimento de padrões vocais; tal programa deve, pois, ser inserido antes desse retorno ao início do programa.

CONCLUSÃO

Verificou-se, no decorrer do trabalho, ser desnecessário o emprego de DMA (hipótese essa que foi inicialmente levantada), uma vez que este só se justifica para taxas de transferência superiores a 10000 bit/s; portanto, com o AD571 a transferência via CPU atende plenamente. Para aplicações que exijam maior velocidade de cálculo, recomenda-se o uso de microprocessadores mais rápidos, como o Z80H. No decorrer do projeto estudou-se ainda a possibilidade de se usar o microprocessador 8086. Ele teria especial vantagem pelo fato de possuir instruções do tipo MUL e DIV que fazem multiplicações e divisões de 16 bits diretamente, bem como um poderoso conjunto de instruções multibyte que indubitavelmente tornariam nosso programa bem mais rápido. Uma utilização ainda mais interessante do 8086 incluiria sua configuração em multiprocessamento (aterrando-se seu pino MN/$\overline{MX}$) com um co-processador de ponto flutuante e uma supervisão de controle por um arbitrador de barramentos.

BIBLIOGRAFIA

ALBRECHT et alii - Electronics Engineer's Handbook - McGraw-Hill
BRIGHAM, E.O. - The Fast Fourier Transform - Prentice Hall
ABRAMOWITZ & STEGUN - Handbook of Mathematical Functions - Dover
LEVENTHAL, Lance A. - Z80 Assembly Language Programming - Osborne-McGraw-Hill
LOGAN, I. - ZX81 ROM Disassembly
RUSSEL & RECTOR - 8086 Handbook - Osborne-McGraw-Hill
UNIVERSITY OF TEXAS - Practical Use of FFT

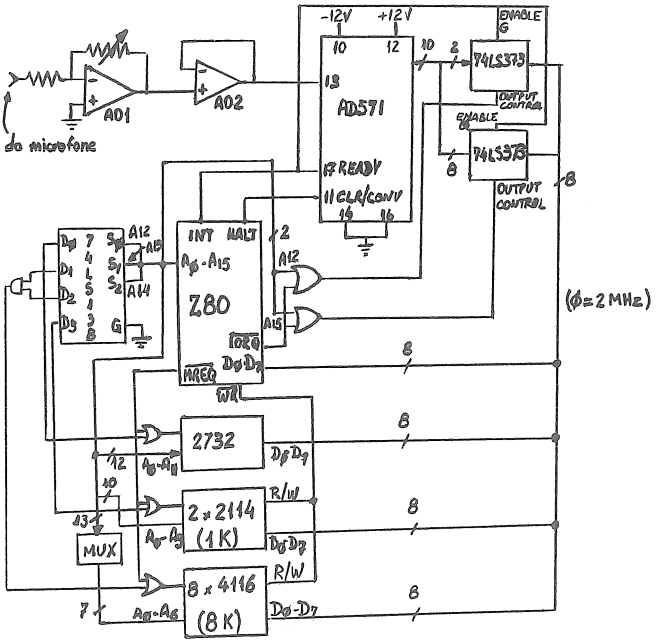


FIGURA 1

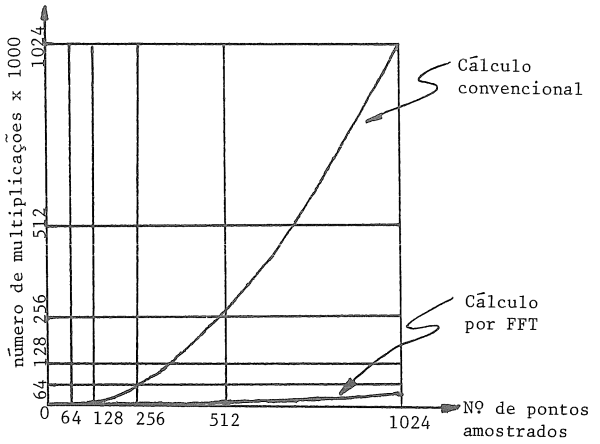


FIGURA 2

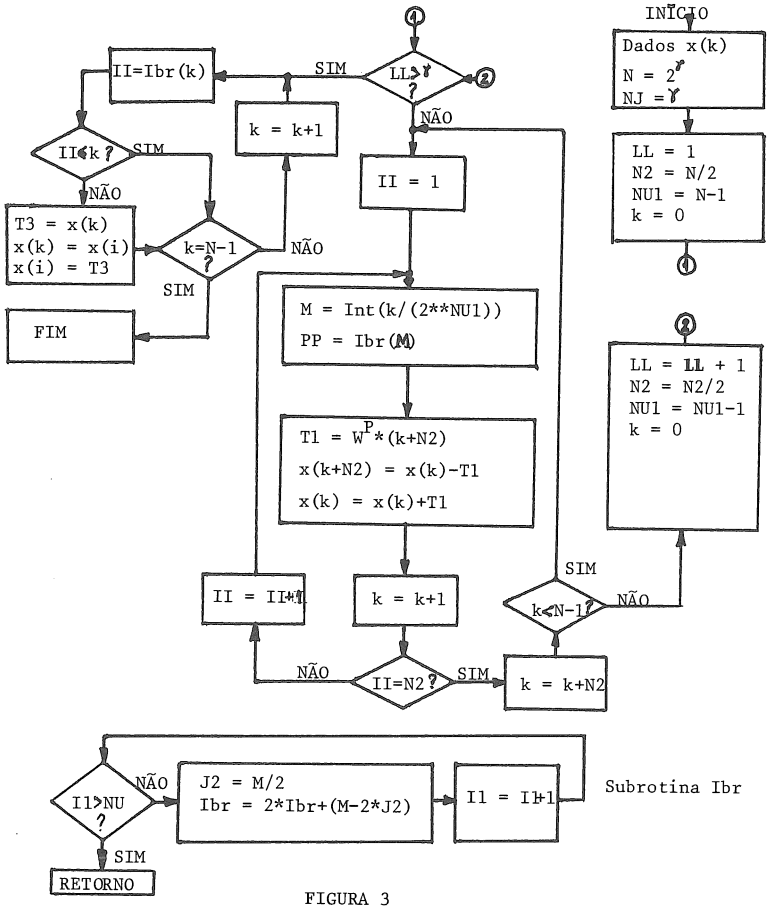


FIGURA 3

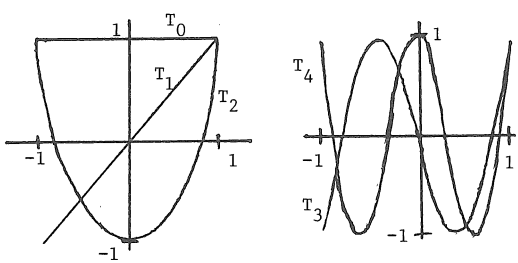


FIGURA 4

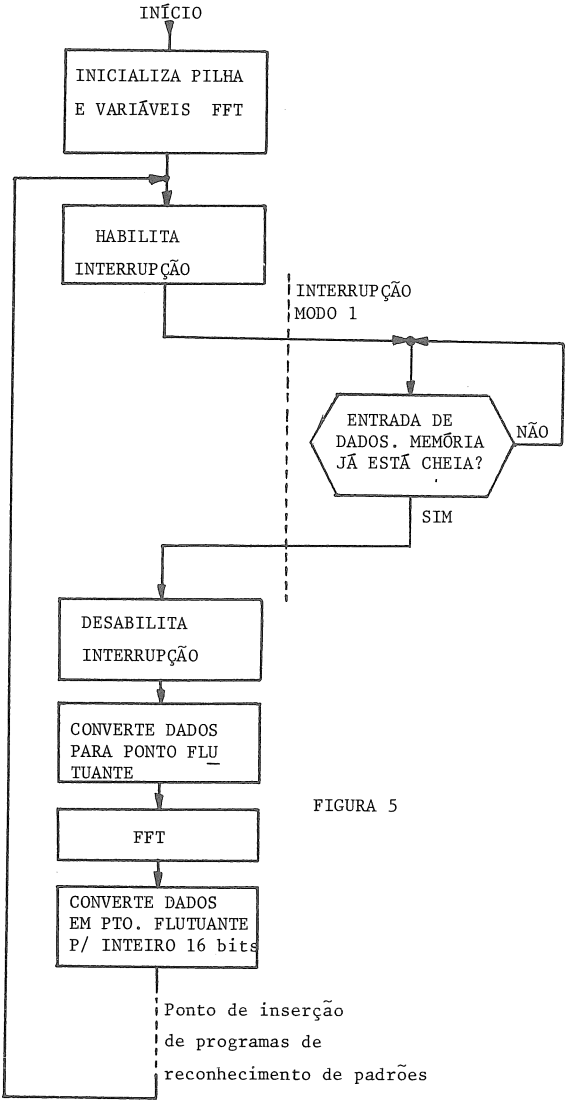

INÍCIO
INICIALIZA PILHA
E VARIÁVEIS FFT
HABILITA
INTERRUPÇÃO
INTERRUPÇÃO
MODO 1
ENTRADA DE
DADOS. MEMÓRIA
JÁ ESTÁ CHEIA?
NÃO
SIM
DESABILITA
INTERRUPÇÃO
CONVERTE DADOS
PARA PONTO FLU
TUANTE
FFT
CONVERTE DADOS
EM PTO. FLUTUANTE
P/ INTEIRO 16 bits
Ponto de inserção
de programas de
reconhecimento de padrões

FIGURA 5

M. H. SZWARCMAN *, A. NESTOROV **

SUMARIO

Este trabalho descreve um microcomputador dedicado que permite o compartilhamento de impressoras entre diversos microcomputadores de uso geral. O sistema inclui ainda caracterfsticas de spooler, permitindo o armazenamento em memória de dados a serem impressos. E uma soluçãO simples e barata para ambientes com vários microcomputadores próximos que necessitam compartilhar recursos de impressão.

ABSTRACT

This paper describes a dedicated microcomputer that allows sharing printers among several general purpose microcomputer systems. The system includes some spooling features, like storage in memory of data to be printed. It is a simple and inexpensive solution for an environment where nearby microcomputers have to share printing resources.

* Engenheiro Eletrônico (PUC/RJ, 1975), M.Sc. Engenharia Elétrica (PUC/RJ, 1979); arquitetura de computadores; Professor do Departamento de Engenharia Elétrica, Pontiffcia Universidade Católica do Rio de Janeiro, Rua Marquês de São Vicente, 225 - Gávea - Rio de Janeiro, RJ - CEP 22453

** Engenheiro Eletrônico (PUC/RJ, 1984), mestrando em eletrônica; Departamento de Engenharia Elétrica, Pontiffcia Universidade Católica do Rio de Janeiro, Rua Marquês de São Vicente, 225 - Gávea - Rio de Janeiro, RJ - CEP 22453

INTRODUÇÃO

O compartilhamento de recursos de hardware e software tem motivado nos últimos anos o desenvolvimento de redes de computadores. A evolução da tecnologia e consequente redução dos custos de hardware tem permitido a interligação de microcomputadores de aplicação geral, não muito distantes entre si geograficamente, através de uma rede de computadores. Este tipo de rede, onde os computadores e periféricos estão próximos é conhecida como rede local. Entretanto, a interligação de computadores através de uma rede local exige um hardware cujo custo e complexidade muitas vezes se compara aos próprios computadores a serem interligados. Em determinadas aplicações, a necessidade de apenas compartilhar um único recurso não justifica o uso de uma rede local.

Um ambiente típico onde as considerações anteriores se aplicam é nos laboratórios de microcomputadores das Universidades. O custo dos periféricos aliado aos poucos recursos financeiros disponíveis impedem a manutenção de laboratórios de ensino de computação com os microcomputadores em uma configuração adequada.

Seria desejável, por exemplo, que cada microcomputador tivesse uma impressora, permitindo ao aluno trabalhar de forma mais eficiente. Porém, a quantidade de impressoras necessárias é proibitiva, como é proibitivo também o custo de instalação de uma rede local apenas para compartilhar impressoras. No trabalho aqui apresentado buscou-se uma solução simples e barata que permitisse o compartilhamento de uma impressora de forma prática e eficiente.

ARQUITETURA DO SISTEMA

Projetou-se então um sistema de computação dedicado, baseado em microprocessador, com arquitetura convencional. Ao invés de apenas multiplexar impressoras entre microcomputadores, decidiu-se construir um sistema que incluísse características de spooler, permitindo principalmente o armazenamento de dados em algum tipo de memória para posterior impressão. Como uma das principais preocupações era o custo, isto é, pretendia-se construir um sistema barato, eliminou-se de início a utilização de discos magnéticos de qualquer espécie para o armazenamento de dados, pelo menos na versão inicial. A idéia era utilizar esta versão num dos laboratórios de graduação, onde as listagens em geral são pequenas, e a frequência de utilização das impressoras não é muito alta. Baseado nisto concluiu-se que seria viável a utilização de memória RAM para o armazenamento temporário dos dados recebidos. Isto de certa forma

impôs a necessidade de contruir um sistema com uma capacidade de endereçamento de mais de 64K bytes.

Uma outra preocupação era com a forma de conexão dos microcomputadores ao sistema. A idéia era utilizar uma interface bastante geral, no sentido de que praticamente qualquer microcomputador pudesse ser conectado ao sistema com um mínimo ou até nenhuma adaptação. Assim, escolheu-se como forma de conexão uma porta serial padrão RS 232-C, disponível em praticamente qualquer microcomputador de uso geral.

Quanto a conexão de impressoras, o sistema deveria suportar as interfaces mais usadas pelas impressoras existentes no mercado, em geral interfaces seriais padrão RS 232-C ou interfaces paralelas do tipo CENTRONICS.

DESCRIÇÃO DO HARDWARE

O projeto é baseado na UCP 8086 da Intel. A escolha deste processador deveu-se aos seguintes motivos: capacidade de endereçamento de 1M byte; estrutura interna adequada ao software que se planejava implementar, com segmentação de memória; disponibilidade e custo. O sistema suporta até 256K bytes de memória RAM dinâmica exapndíveis até 512K bytes. Foi reservado ainda, um espaço de 64K bytes para EPROM.

O sistema dispõe de 10 interfaces seriais padrão RS 232-C, para conexão de microcomputadores e impressoras. A definição dos canais onde estarão conectados microcomputadores e dos canais onde estarão conectadas impressoras seriais é feita por software. No que diz respeito ao hardware, qualquer canal serial pode operar conectado a uma impressora ou a um microcomputador, indistintamente. As velocidades de operação de cada canal são independentes, isto é, os canais podem operar em velocidades diferentes entre si e a velocidade máxima de operação de cada canal é de 9600 bps. A configuração das velocidades de operação também é feita por software. O controle de fluxo de dados para as impressoras seriais pode ser feito por hardware, através dos sinais RTS e CTS, ou por software, usando protocolo XON/XOFF. Existem ainda 2 portas paralelas para conexão de impressoras com interface do tipo CENTRONICS ou similar. O hardware suporta toda entrada e saída por interrupção.

DESCRIÇÃO DO SOFTWARE

O software do sistema é todo residente, e fica armazenado em EPROM. A primeira versão, implementa apenas as funções básicas necessárias ao funcionamento do sistema. Estas funções incluem recepção dos dados,

alocação de segmentos, gerenciamento da fila de impressão e envio de dados para as impressoras. A principal preocupação no projeto do sofware foi no sentido de aproveitar eficientemente os recursos da máquina, principalmente no que diz respeito a memória disponível para o armazenamento de dados a serem impressos. Num sistema de spool "ideal", a capacidade de armazenamento deve ser infinita, de forma a eliminar o tempo de espera do usuário.

Assim, a memória é dividida em segmentos, que vão sendo alocados a medida das necessidades de um canal. O tamanho de cada segmento pode ser definido na configuração do sistema (o protótipo utiliza segmentos de 4K bytes). As informações de controle de alocação de segmentos são mantidos numa tabela associada a cada processo de impressão e incluem: lista de segmentos alocados, tamanho da lista, número do segmento corrente e apontador de próxima posição livre no segmento corrente. Uma política de alocação de segmentos que dá prioridade ao canal que tem o menor número de segmentos alocados. Isto garante uma distribuição justa dos segmentos aos diversos canais ativos. Um protocolo simples, que consiste apenas do envio de um byte de controle indicando fim de envio de dados, controla a transferência de dados entre o microcomputador e o servidor.

A impressão dos dados recebidos por um canal é iniciada assim que o programa detecta uma impressora disponível. Isto significa que a impressão de dados recebidos pode ser iniciada antes que o envio destes dados tenha sido completado pelo microcomputador, desde que os dados enviados já tenham preenchido no mínimo 1 segmento. Quando não há impressora disponível e o microcomputador completa o envio de dados - através do envio do byte de controle - os dados recebidos são enfileirados para impressão e o canal é liberado, podendo receber um novo conjunto de dados. É possível ainda selecionar uma determinada impressora através do envio de uma sequência de controle encabeçando os dados. Quando esta sequência é omitida o programa aloca a primeira impressora disponível.

Para facilitar a depuração de hardware e software foi implementado um programa monitor. Este monitor possui uma série de funções de auxílio a depuração como por exemplo, execução de programas com inserção automática de pontos de parada, execução passo a passo de programas e carregamento de programas através de uma das interfaces seriais. A interação com o monitor é feita usando um terminal acoplado a uma das interfaces seriais. O monitor foi uma ferramenta bastante útil, agilizando bastante o trabalho de depuração, tanto do hardware quanto do software. O monitor é

mantido residente para testes de novas versões e serviços de manutenção.

CONCLUSÃO

Foi montado um primeiro protótipo do servidor, com 128k bytes de RAM e 8k bytes de EPROM que foi testado no laboratório de graduação de microprocessadores da PUC/RJ, acoplado a uma impressora de 100 cps e 6 microcomputadores. Devido ao baixo volume de impressão, tipicamente exigido pelos trabalhos deste laboratório, obteve-se resultados excelentes, isto é, a demanda foi plenamente atendida pelo servidor. Cada microcomputador possuia virtualmente uma impressora.

Testes simulando ambientes com volume maior de impressão e com duas impressoras conectadas ao servidor - uma de 300 lpm e uma de 100 cps - mostraram a viabilidade do uso do sistema nestes ambientes.

O desempenho do sistema está diretamente ligado ao número de impressoras conectadas, a velocidade destas impressoras e a quantidade de memória disponfvel. Acredita-se, por exemplo, que utilizando o máximo de memória possfvel e impressoras rápidas consiga-se um bom desempenho em ambientes com volume de impressão elevado. A flexibilidade do sistema, no que diz respeito à configuração e ao tipo de impressora suportada, permite uma adaptação a ambientes com diferentes demandas de impressão.

BIBLIOGRAFIA

Madnick, S. E., Donovan, J. J., "Operating Systems", McGraw-Hill, 1974.

Rector, R., Alexy, G., "The 8086 Book", Osborne/McGraw-Hill, Berkeley, 1980.

"Microsystem Components Handbook", Intel, 1984.

"Memory Components Handbook", Intel, 1984.

"iAPX 86/88, 186/188 User's Manual", Intel 1983.

"Z-80 SIO Technical Manual", Zilog, 1978.

TRADUCTOR DE UN LENGUAJE ESTRUCTURADO A DESCRIPCION ELEMENTAL DE HARDWARE

H. ANTILLANCA E.* R. VALLEJOS C**

RESUMEN

Se diseñó un traductor, cuyo código fuente está escrito en un lenguaje de alto nivel, apropiado para describir 'hardware', y el código objeto generado es el listado de los circuitos e interconexiones de una máquina digital, que funcionalmente resuelve el mismo problema del programa fuente.

También se diseñó el lenguaje de programación. Esto se hizo mediante modificaciones al lenguaje Pascal -escrito en castellano- de forma tal que permitiera incorporar las estructuras de datos propias de los circuitos digitales.

La máquina digital, obtenida automáticamente por el traductor, está basada en una unidad de control microprogramada. Los microprogramas necesarios también son generados por el traductor.

ABSTRACT

A translator was designed, such that its source code is written in a high -level language, appropriate for "hardware" description, and its generated object code is the printout, or listing, of the circuits and interconnections of a digital machine which functionally solves the same problem of the source program.

Furthermore, the programming language was designed. This was done by modifying the Pascal language - written in Spanish - in such a form as to permit the incorporation of the digital circuits proper data structures.

The digital machine automatically obtained by the translator, is based upon a microprogrammed control unit. The necesary microprograms are also generated by the translator.

* Ingeniero en Electrónica (UCV, Chile, 1980), egresado de Magister en Sistemas Digitales (UTFSM, Chile, 1983). Universidad Técnica Federico Santa María, Departamento de Electrónica, Casilla 110-V, Valparaíso-CHILE.

** Ingeniero Civil Electrónico (UTFSM, Chile, 1976). Profesor del Departamento de Electrónica, Universidad Técnica Federico Santa María, casilla 110-V, Valparaíso-CHILE.

INTRODUCCION

El diseño de los sistemas digitales ha evolucionado conforme lo ha hecho la tecnolología de los componentes, desde la integración de las compuertas básicas, donde el diseño implicaba usar ecuaciones booleanas y diagramas lógicos, hasta la integración de circuitos lógicos complejos (circuitos MSI/LSI/VLSI) cuya modularidad permite ahora al diseñador escribir las máquinas digitales mediante abstracciones de componentes y Lenguajes de Descripción de Hardware (HDLs).

Así es como ha nacido una gran variedad de HDLs. Muchas publicaciones muestran las discusiones y los esfuerzos realizados por los autores para lograr una uniformidad en pro de especificar claramente una máquina digital en todos los diferentes niveles de representación posibles [1,2,3,4,5,6,7].

En forma paralela al nacimiento de los HDLs se ha agregado, al diseño de los sistemas digitales, el uso de una herramienta poderosa como lo es el diseño apoyado por computador (CAD), y en consecuencia producir ambientes computacionales de diseño [8], que apuntan sus esfuerzos a la simulación [9] y a la síntesis automática de sistemas digitales [10,11].

El presente trabajo aprovecha la evolución ocurrida en los lenguajes de programación de computadores, con el objeto de diseñar máquinas digitales. La idea es incorporar al diseño de hardware los conceptos de modularidad y las metodologías jerárquicas de diseño que posee el diseño de software. Ejemplo claro es el lenguaje de programación Pascal, el cual tiene la ventaja de permitir al usuario diseñar programas estructurados usando sentencias de control de flujo de alto nivel [12,13].

El objetivo ha sido evaluar una técnica de diseño automático de máquinas digitales apoyado por computador, para lo cual se ha construido un traductor capaz de generar una representación elemental de la máquina (alambrado), a partir de una descripción inicial de ésta en términos de un HDL de alto nivel y a la vez estructurado. En consecuencia, diseñar una máquina digital es equivalente a resolver un problema específico escribiendo un programa en ese lenguaje.

1. DESCRIPCION DEL TRABAJO REALIZADO

El trabajo consta de dos partes:

1) Definición de un Lenguaje de Descripción de 'Hardware' de alto nivel con sentencias de control estructuradas, y

2) Construcción de un Traductor.

1.1. El Lenguaje de Descripción de Hardware de Alto Nivel

El HDL diseñado, llamado lenguaje DMD (Descripción de Máquinas Digitales), es una variación del lenguaje de programación de alto nivel llamado Pascal, pero orientado a satisfacer las necesidades de describir sistemas digitales. La sintaxis completa del lenguaje DMD se presenta en el Apéndice A.

El nivel del lenguaje es tal que no se preocupa tanto del control de la máquina sino del aspecto funcional. Esto se hace mediante las sentencias de control de flujo estructuradas. De esta forma, el diseñador se desliga del aspecto control de la máquina y puede, usando las facilidades de modularidad de las sentencias, definir el hardware en base a los requerimientos de software. La parte control es un problema que resuelve el traductor.

El lenguaje DMD exige del diseñador especificar:

a) Los tipos y variables a usar, llamada Parte Declaraciones, y

b) La descripción del programa, llamada Parte Bloque.

1.1.1. Parte Declaraciones

La parte declaraciones considera las declaraciones de tipo y las declaraciones variables.

La declaración de tipo permite definir, sobre ciertos recursos, atributos que posibilitan al traductor chequear la legalidad y compatibilidad de las construcciones realizadas. Así, por ejemplo, en la mayoría de los lenguajes se pueden definir las variables tipo enteros, y cualquier operación en que el resultado deba ser entero y se opere, por ejemplo, con una variable booleana, es ilegal, sin embargo se aceptan ciertas operaciones con variables de tipo real.

En el lenguaje DMD, se puede definir, por ejemplo, variables tipo contador, pero no se puede aplicar sobre ellas operaciones aplicables solo a variables de tipo registro, como por ejemplo operaciones de corrimientos.

El tipo básico del lenguaje DMD es el tipo booleano, sin embargo, no es

declarable explítamente. Este tipo es el átomo que compone todas las variables declarables y, en consecuencia, todas pueden ser tratadas como variables booleanas en su nivel mínimo.

Los tipos básicos declarables explícitamente son aquellos tipos que permiten identificar el recurso de hardware que se asocia a la variable del tipo dado. Estos tipos son: tipo flip-flop, tipo registro, tipo contador, tipo terminal y tipo memoria.

1.1.2. Parte bloque

Corresponde a la parte en que se describe el funcionamiento de la máquina mediante sentencias. Hay dos tipos de sentencias:

a) Las que efectúan el control del programa
b) Las que indican una acción sobre el recurso

Las sentencias de control utilizables son:

- COMIENZO-FINAL
- SI-ENTONCES-SINO
- MIENTRAS-HAGA
- REPITA-HASTA

Las sentencias que indican acción sobre un recurso corresponden a:

- ASIGNACION (Transferencias entre registros)
- PRIMITIVAS (Operaciones sobre un recurso específico)

1.1.3. Ejemplo Ilustrativo

A continuación se presenta un ejemplo que muestra la filosofía de diseñar con DMD:

```
MAQUINA PARIDAD;

(* Este es un comentario.  El traductor lo ignora *)
(* El siguiente programa es la descripción autocontenida de una máquina
simple que determina, en forma serie, la paridad de un registro *).
```

```
TIPOS BIT       : FLIPFLOP;
      CONT      : CONTADOR DE 4 BITS;
      REG       : REGISTRO DE 8 BITS;

VAR   A  : REG;
      CC:  CONT;
      BP:  BIT;
      FIN: BIT;

COMIENZO
A:='3F';
LIMPIE(CC);
LIMPIE(BP);
LIMPIE(FIN);
REPITA
   BP:=BP % A[8]; (* OR EXCLUSIVO ENTRE BP y A[8]*)
   DESPLZ(DER,A,A[8]); (*Desplazamiento cíclico a la derecha de A*)
   INCR(CC);
HASTA {CC='8'};
FIN:=BP;
FINAL.
```

1.2. El Traductor

El Traductor realiza básicamente las siguientes tareas:

a) Revisar la sintaxis del lenguaje DMD.

b) Analizar legalidad y compatibilidad de las operaciones sobre los recursos declarados.

c) Producir el listado con el alambrado de la máquina a nivel de hardware, esto es, los tipos de circuitos integrados necesarios, la cantidad de cada uno de ellos, y las interconexiones entre ellos.

d) Sintetizar operaciones booleanas en hardware.

e) Analizar situaciones que producen repeticiones de alambrado, a fin de evitarlo.

f) Generar el código del microprograma que controla la secuencia de operaciones

de la máquina. Esto se deduce de las sentencias de control de flujo del lenguaje DMD.

1.3. Arquitectura de la Máquina Generada

En el proceso de traducir el lenguaje DMD, es tarea del traductor generar la estructura que produce el control de la máquina descrita. La estructura generada consta de una unidad de control de microprograma (secuenciador), que controla la secuencia de obtención de las microinstrucciones de una memoria, y la memoria que almacena el microprograma que controla la máquina descrita. Esta estructura es lo único que se mantiene igual para todas las máquinas generadas, no así el microprograma que depende de los algoritmos de funcionamiento de las mismas.

La arquitectura de la máquina generada, que se observa en la figura 1, se puede descomponer en tres estructuras de hardware: la estructura de control -ya descrita-, la estructura objeto -sobre la cual se ejecuta el control-, esto es, aquellos recursos dependientes de las variables declaradas por el diseñador, y la estructura de interfaz entre la estructura de control y la estructura objeto -estructura compuesta por elementos de hardware que se asocian a la máquina objeto para que se haga efectivo el control sobre ella.

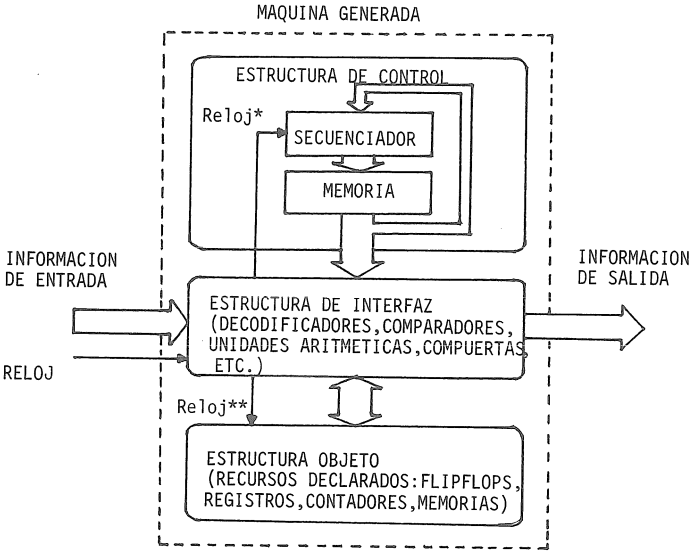


Fig. 1. Arquitectura de la Máquina Generada.

2. CONCLUSIONES

El uso del lenguaje DMD ha permitido profundizar, desde el punto de vista del diseño, en una conceptualización de los sistemas digitales. Con este lenguaje es posible enfrentar el diseño de hardware de la misma manera como se enfrenta el diseño de software, esto es, emplear las metodologías de diseño estructurado y 'Top-Down', permitiendo al diseñador desligarse de detalles estructurales de la máquina, y concentrarse fundamentalmente en los aspectos algorítmicos y de almacenamiento de la información.

3. REFERENCIAS

[1] M.R. Barbacci: "A Comparison of Register Transfer Languages for Describing Computer and Digital System", IEEE Trans., 1975, c-24, pp.137-150.

[2] G.J. Lipovski: "Hardware Description Languages: Voices from the Tower of Babel", Computer, 1977, 10, pp. 14-17.

[3] Harry F. Jordan and Burton J. Smith: "The Assignment Statement in Hardware Description Languages", Computer, 1977, 10, pp. 43-49.

[4] G.R. Peterson: "Teaching Digital System Design with a Hardware Description Languages", Proc., 1978 Frontier in Education Conference, ASEE/IEEE.

[5] "Instruction Set Processor Specifications (ISPS): The Notation and its Application", IEEE Trans. on Comput., Vol. c-30, Jan. 1981.

[6] C.A.G. Enstone, "HARTRAN: A language for Top-Down Digital System Design", GEC Hirst Research Centre, Wembley, Middlesex, England, 1981.

[7] Ed Thompson, A. Lekkos, Don Ross, R. von Blucher, "TEGAS Design Language (TDL) - A System for Modular Description and Top-Down/Bottom-Up Verification of LSI and VLSE". C. Computing System and Service, Inc., Austin, Texas, 1980.

[8] P.W. Foulk, R.A. Mason, "A Design Enviroment for Digital Hardware", Computer Engineering, Heriot-Watt University, Edinburgh, 1980.

[9] Melvin A. Breuer, Editor: "Digital System Design Automation: Language, Simulation & Data Base", Computer Science Press, Inc., Woodland Hills, California, 1975.

[10] S.W. Director, A.C. Parker, D.P. Siewiorek, and D.E. Thomas: "A Design Methodology and Computer Aids for Digital VLSI System", IEEE Trans. Circuits System, Vol. cas-28, July, 1981.

[11] Donald E. Thomas: "The Automatic Synthesis of Digital System", Proceeding of the IEEE, Vol. 69, October 1981.

[12] Dahl, Dijkstra and Hoare: "Structured Programming", Academic Press, 1978.

[13] Niklaus Wirth: "Algorithms+Data Structure=Program", Prentice-Hall, Inc.

[14] G.M. Baudet, M. Cutler, M. Davio, A.M. Peskin, F.J. Ramming: "The Relationship between HDLs and Programming Languages". Silver Spring, MD, USA: IEEE Comput. Soc. Press. Inc., 1983.

APENDICE A

SINTAXIS DEL LENGUAJE DMD (DESCRIPCION DE MAQUINAS DIGITALES)

El formalismo BNF (Bankus Naur Form) se usa a continuación para describir la sintaxis del presente lenguaje.

<Descripción de Máquina Digital>::=<encabezado de máquina> <bloque>

<encabezado de máquina>::=MAQUINA <identificador de máquina> ;
<identificador de máquina>::=<identificador>
<identificador>::=<letra>{<letra>|<dígito>}
<letra>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<dígito>::=Ø|1|2|3|4|5|6|7|8|9

<bloque>::=<declaraciones><sentencia compuesta>

<declaraciones>::={<declaración de const>}
{<declaración de tipos>}
{<declaración de variables>}
{<declaración de submáquinas>}

<declaración de const>::=CONST <definición de const>{;<definición de const>};
<definición de const>::=<identificador de const>=<hexadecimal>
<identificador de const>::=<identificador>
<hexadecimal>::='<hexdígito>{<hexdígito>}'
<hexdígito>::=<dígito>|A|B|C|D|E|F

<declaración de tipos>::=TIPOS<definición de tipo>{;<definición de tipo>};
<definición de tipo>::=<identificador de tipo>=<tipo>
<identificador de tipo>::=<identificador>
<tipo>::=<tipo simple>|<tipo estructurado>
<tipo simple>::=<tipo básico>|<identificador de tipo>
<tipo básico>::=<flipflop>|<registro>|<contador>|<terminal>|<memoria>
<flipflop>::=FLIPFLOP
<registro>::=REGISTRO DE <dimensión del ancho> BITS
<contador>::=CONTADOR DE <dimensión del ancho> BITS
<terminal>::=TERMINAL DE <dimensión del ancho> BITS
<memoria>::=MEMORIA DE <dimensión del ancho>*<dimensión del largo> BITS
<tipo estructurado>::=<tipo arreglo>|<tipo record>
<tipo arreglo>::=ARREGLO [<entero>..<entero>] DE <tipo>

<entero>::=<dígito>{<dígito>}
<tipo record>::=RECORD <lista de campos> FINAL
<lista de campos>::=<definición de campo>{;<definición de campo>}
<definición de campo>::=<identificador de campo>:<tipo>

<declaración de variables>::=VAR <definición de var>{;<definición de var>};
<definición de var>::=<identificador de var>{,<identificador de var>}:<tipo>

<declaración de submáquinas>::=<definición de submáq>{;<definición de submáq>};
<definición de submáq>::=<encabezado de submáquina><bloque>
<encabezado de submáquina>::=SUBMAQ <identificador de submáquina>;
<identificador de submáquina>::=<identificador>

<sentencia compuesta>::=COMIENZO <sentencia>{;<sentencia> FINAL
<sentencia>::=<asignación>|<identificador de submáquina>|<primitiva>|

<sentencia si-entonces-sino>|<sentencia mientras-haga>|
<sentencia repita-hasta>|<sentencia con-haga>|
<sentencia desde-hasta-haga>|
<sentencia compuesta>

<asignación>::=<variable>:=<expresión>
<variable>::=<identificador de var><designador>
<designador>::={.<identificador de campo> [<dirección>]}
<dirección>::=<variable>:<entero>
<expresión>::=<constante>|<expresión simple>|<expresión booleana>
<constante>::=<identificador de constante>:<hexadecimal>
<expresión simple>::=<primer operando><operador><segundo operando>
<primer operando>::=<variable>|~<variable>
<segundo operando>::=<variable>|~<variable>
<operador>::=+|-|#|%|&
<expresión booleana>::=<función booleana>|<parizq><comparación><parder>
<función booleana>::=<término booleano>{#<término booleano>}
<término booleano>::=<subtérmino booelano>{%<subtérmino booleano>}
<subtérmino booleano>::=<factor booleano>{&<factor booleano>}
<factor booleano>::=<variable>|(<expresión booleana>)|~<factor booleano>
<parizq>::={
<parder>::=}
<comparación>::=<primer comparando><relación><segundo comparando>
<primer comparando>::=<variable>
<segundo comparando>::=<variable>:<constante>
<relación>::=>|<|=

```
<primitiva>::=<cuenta>|<desplaz>|<colocación>
<cuenta>::=<ident.primitiva cuenta>(<variable>)
<ident.primitiva cuenta>::=INCR | DECR
<desplaz>::=<ident.primitiva desplaz>(<sentido>,<variable>,<desde>)
<ident.primitiva desplaz>::=DESPLZ
<sentido>::=DER | IZQ
<desde>::=<dato simple>
<dato simple>::= Ø | 1 | <expresión booleana>
<colocación>::=<id. prim.coloc.>(<variable>)
<ident.primitiva coloc>::=LIMPIE PONGA

<sentencia si-entonces-sino>::=SI <expresión booleana>
                               ENTONCES <sentencia>
                               {SINO <sentencia>}
<sentencia mientras-haga>::=MIENTRAS <expresión booleana>
                            HAGA <sentencia>
<sentencia repita-hasta>::=REPITA <sentencia> HASTA <expresión booleana>

<sentencia con-haga>::=CON <variable> HAGA <sentencia>

<sentencia desde-hasta-haga>::=DESDE <asignación> HASTA <constante>
                              HAGA <sentencia>
```

SUPORTE PARA DESENVOLVIMENTO DE SISTEMAS DISTRIBUIDOS BASEADO NUM BARRAMENTO PARALELO CENTRALIZADO

C. KIRNER*

E. MARQUES**

SUMÁRIO

Este artigo apresenta o projeto de um sistema distribuido baseado num barramento paralelo centralizado. Descreve-se a arquitetura do sistema e analisa-se as questões de projeto do barramento paralelo e do processador de comunicação.

ABSTRACT

This paper presents the design of a distributed systems based in a centralized parallel bus. The architecture of the systems is described and the design of the bus and communication processor is analysed.

(*) Engenheiro Eletricista(EESC-USP, 1973), Mestre em Engenharia Eletrônica(ITA, 1978) Doutorado em Engenharia de Sistemas e Computação (COPPE-UFRJ, em andamento), Sistemas Operacionais, Arquitetura, Redes. Prof. Assistente. DCEs-UFSCar - C.P. 384. 13560 - São Carlos - S.P. - Fone (0162) 71-1100 - Ramal 143.

(**) Bacharelando em Ciência da Computação (UFSCar - 1985), Bolsista de Iniciação Científica do CNPq em 1984 e 1985;
Arquitetura, Redes, Sistemas Operacionais. DCEs - UFSCar;
C.P. 384 - 13560 - São Carlos - S.P. - Fone (0162) 71-1100 - Ramal 143.

1. INTRODUÇÃO

Um grupo de pesquisa do Departamento de Computação e Estatística da UFSCar investiu, durante 2 anos, no desenvolvimento de um subsistema de comunicação com conexão ponto a ponto [6] para servir como parte de um suporte para o desenvolvimento de Sistemas Distribuídos. O fator determinante na concepção do projeto foi única e exclusivamente o interesse acadêmico.

Um dos objetivos do projeto era o de gerar conhecimento e motivação para que o grupo, ou mesmo membros do grupo, passassem a atuar na área de redes e sistemas distribuídos, de forma mais decisiva, com um enfoque que permitisse, ao mesmo tempo, uma especulação científica de vanguarda e uma proximidade com as necessidades atuais e futuras do mercado.

Assim, há pouco mais de 1 ano, motivados pela análise de dois tipos de rede que enfatizava a proposta de uma rede estrela coincidente [7], passamos a trabalhar na proposta de um barramento paralelo centralizado. Para isso contamos com o auxílio do CNPq que, através de uma bolsa de iniciação científica, propiciou as condições para o estudo de alguns barramentos e a elaboração, em linhas gerais, da proposta do barramento paralelo centralizado e de suas interfaces de acesso [8].

O subsistema de comunicação, composto de vários processadores de comunicação, interligados por um barramento paralelo centralizado ("backplane"), baseia-se na existência de "buffers" de emissão e de recepção ligados às linhas paralelas de alta velocidade de transmissão. Essa característica faz com que o tempo de comunicação entre dois processadores de comunicação, já preparados, seja muito pequeno, ou praticamente nulo, restando assim o tempo de preenchimento do buffer emissor e o tempo de esvaziamento do buffer receptor, perfeitamente previsíveis, de forma que o tempo total de comunicação seja bem determinado.

Além disso procurou-se usar linhas redundantes, no barramento paralelo, com capacidade para substituição de algumas partes, proporcionando tolerância à falha com capacidade de reconfiguração das linhas, em alguns casos.

Estas duas características, ou seja, tempo determinístico na comunicação e tolerância à falha recomendam a utilização desse tipo de subsistema na solução de problemas de aplicação em tempo real, particularmente os de controle de processo, mesmo a um custo mais alto, o que normalmente acontece nesses casos.

O projeto visa gerar a infraestrutura necessária ao estudo de primitivas de comunicação e sincronização para sistemas de aplicação em tempo real, e ao desenvolvimento de sistemas operacionais distribuídos.

Outro aspecto, a ser destacado, consiste na possibilidade de comparação de uso, em sistemas distribuídos, de conexão ponto a ponto e de conexão por barramento paralelo centralizado, dos pontos de vista de implementação, desempenho, resistência à

Cabe citar que o projeto está sendo desenvolvido com o uso de microprocessadores, onde estiver sendo possível, aproveitando-se as vantagens de custo e flexibilidade desses componentes, mas tomando-se todo o cuidado para que a estruturação do projeto permita a utilização de estruturas mais rápidas e mais adequadas a determinadas exigências de aplicações especiais.

O uso de barramento centralizado já vem sendo explorado, especialmente com o esquema de contenção de barramento, conforme publicação recente [2].

2. ARQUITETURA DO SISTEMA

O sistema constitui-se de um subsistema de comunicação composto por vários *processadores de comunicação* (PC), interligados através de um barramento paralelo disposto num "backplane", e de um conjunto de processadores de trabalho, denominados processadores operadores (PO), conforme a figura 1.

O subsistema de comunicação será montado em um gabinete, contendo uma fonte confiável ("no break"), o barramento paralelo com 72 linhas dispostas no painel trazeiro e 64 placas de circuito ligadas ao barramento, cada uma correspondendo a um processador de comunicação. Este conjunto poderá ser colocado dentro de uma sala bem protegida, de onde sairão todas as ligações para os processadores operadores remotos e, eventualmente, para dispositivos periféricos, sensores e atuadores que estiverem ligados a processadores de comunicação expandidos. Cabe salientar que um processador de comunicação expandido conte

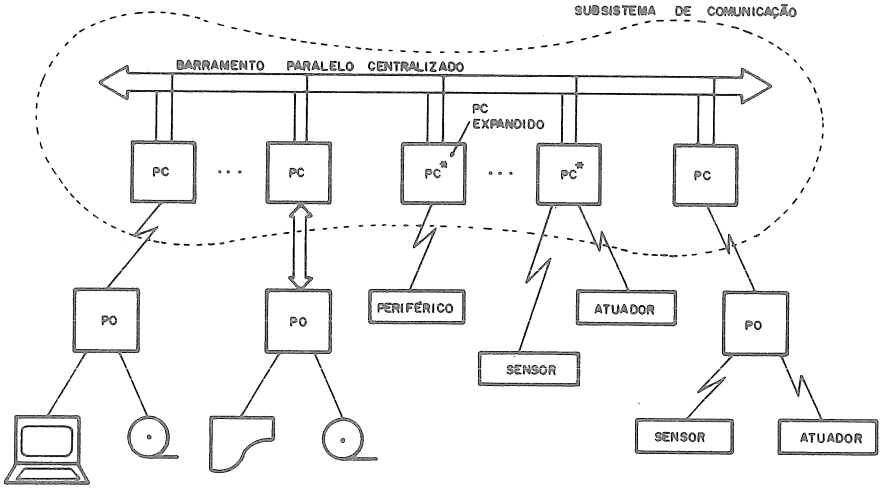


Figura 1 - Configuração Genérica do Sistema Distribuído

rá, além das funções de um processador de comunicação normal, funções simples de controle de alguns periféricos, tais como terminal de vídeo e impressora, e funções de leitura de dispositivos sensores e acionamento de dispositivos atuadores. A limitação, a 64 processadores de comunicação num barramento, deve-se às restrições de componentes eletrônicos "tri-state" e à necessidade de manter-se o comprimento do barramento a dimensões reduzidas. A centralização do subsistema de comunicação, além de facilitar a manutenção, contribui bastante para o aumento da segurança do sistema, uma vez que os maiores riscos de ruptura, incidindo com maior frequência nos cabos externos, comprometerão somente a comunicação individual de cada processador operador com o subsistema, sem afetar sensivelmente o funcionamento do conjunto.

A comunicação entre processadores operadores ocorrerá através dos processadores de comunicação correspondentes e do barramento paralelo centralizado de alta velocidade, que é o meio compartilhado de comunicação. A alta taxa de transferência de informação entre processadores de comunicação será obtida muito mais pela existência de várias linhas paralelas, do que pela velocidade de cada linha. Para permitir que todos os processadores de comunicação utilizem o barramento, é necessário que cada um utilize-o no menor tempo possível, sem monopolizá-lo, e para isso é necessário a existência de "buffers" de emissão e "buffers" de recepção em cada processador de comunicação. Esses "buffers" (figura 2) servirão para a montagem da informação, a ser transferida, ou para a sua desmontagem para posterior utilização, ou retransmissão, em taxas menores, para um processador operador específico.

Um processador operador emissor fará a transferência de informação para um processador operador receptor da seguinte forma. Primeiro, o processador operador emissor transferirá a informação para o processador de comunicação origem, a ele ligado, por bit ou por byte, dependendo da ligação ser serial ou paralela. Esse processador de comunicação irá coletando as partes da informação, conforme a sua chegada, e montando-as convenientemente no "buffer" de emissão. Quando a montagem terminar, o processador de comunicação origem acionará o seu circuito de disputa do barramento e, tão logo ganhe acesso, verificará se o processador de comunicação destino está pronto para receber a informação ("buffer" de recepção vazio). No caso do processador de comunicação destino não encontrar-se pronto, o processador de comunicação origem desistirá do barramento e aguardará um tempo para nova tentativa. No caso do processador de comunicação destino estar pronto, então haverá a transferência da informação, do "buffer" de emissão do processador de comunicação origem, para o "buffer" de recepção do processador de comunicação destino (figura 2). Este último processador encarregar-se-á de desmontar a informação recebida e de transferí-la ao processador operador, a ele ligado, num padrão de transferência compatível com a ligação.

O tempo máximo de comunicação entre dois processadores operadores, no pior caso, ou seja, quando n processadores estiverem competindo pelo barramento, e esses dois

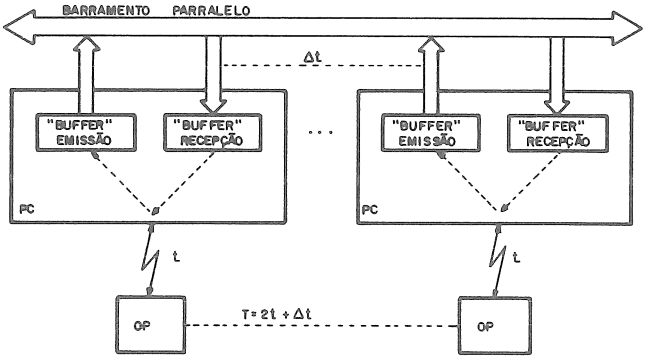


Figura 2 - Comunicação entre 2 Processadores Operadores

forem atendidos por último, deverá ser a soma: tempo gasto na transmissão PO-PC e no preenchimento do "buffer" de emissão mais o tempo de esvaziamento do "buffer" receptor e respectiva transmissão PC-PO mais n vezes o tempo de transferência de um "buffer" pelo barramento, correspondendo ao tempo de espera mais o tempo de transferência. Como o tempo de transferência de um "buffer" pelo barramento deverá ser da ordem de centenas ou milhares de vezes menor do que o tempo gasto na transmissão PO-PC e no preenchimento do "buffer", ou vice-versa, ele terá pouca influência no tempo total de comunicação entre processadores operadores, com a ressalva de que os processadores de comunicação sempre deverão estar prontos. Sabendo-se que esta ressalva, muitas vezes, não corresponde a realidade, a garantia de tempo bem definido de comunicação desaparecerá, a menos que se garanta, a nível de modelamento do sistema, que nunca ocorrerá tentativa de transferência de "buffer" para um processador de comunicação ocupado, ou se consiga formas de contornar o problema. Uma dessas formas consiste na introdução, em cada processador de comunicação, de "buffers" adicionais em número suficiente para armazenar todas as transferências, que possam ocorrer dentro de curtos períodos de tempo, para serem atendidas de acordo com uma ordem de prioridade. Nessa situação, o tempo máximo de comunicação entre 2 processadores operadores poderá variar, dependendo das prioridades envolvidas, mas dentro de valores perfeitamente previsíveis.

É interessante notar, em toda essa análise, que é quase impossível que um processador de comunicação monopolize o uso do barramento ou impeça a transferência de informação de processadores de comunicação de baixa prioridade, ao tentar transferir seqüências de informação, pois o preenchimento de "buffer", que é uma atividade bem mais demorada do que a transferência pelo barramento, propiciará o intervalo de tempo necessário para as outras transferências pendentes.

3. PROJETO DO BARRAMENTO PARALELO

O barramento paralelo foi concebido inicialmente para ter 72 linhas,sendo 40 para transferência de informação (pacotes de mensagem), 4 para atuar como reserva das linhas de informação, 8 para endereço origem, 8 para endereço destino, e as 12 restantes para controle (figura 3).

As linhas de controle, além de indicar o estado do barramento, servirão também: para implementar o protocolo necessário à transferência de informações; para alterar o padrão de transferência; e para realizar testes e ajustar o subsistema de comunicação. O estado do barramento poderá ser verificado através das linhas *desocupado* e *er*

Nº DE LINHAS	NOME	
32	dados	- transferir a informação útil de um pacote.
8	cabeçalho	- transferir o cabeçalho de um pacote.
4	reserva	- servir de linhas sobressalentes para as linhas de transferência de um pacote.
8	endereço origem	- permitir a disputa pelo acesso ao barramento e indicar a origem do pacote a ser transferido.
8	endereço destino	- indicar qual processador de comunicação irá receber o pacote.
1	desocupado	- indicar se o barramento está ocioso.
1	$\overline{\text{clock-in}}$	- disparar o carregamento do pacote no processador de comunicação destino.
1	$\overline{\text{reset}}$	- colocar todos os processadores de comunicação na condição inicial.
1	$\overline{\text{desconectar}}$	- isolar processador de comunicação com falha
1	$\overline{\text{erro-tensão}}$	- indicar tensão no barramento fora de especificação.
1	$\overline{\text{limpo-para envio}}$	- indicar que o processador de comunicação destino está pronto para receber o pacote.
1	$\overline{\text{teste}}$	- solicitar a execução das rotinas de teste.
1	modo-transm	- indicar transferência de 8 ou de 32 bits.
2	reformul.	- indicar qual dos 4 bytes das linhas de dados estará sendo usado.
1	$\overline{\text{não-autor.}}$	- indicar que o processador de comunicação destino não está pronto para recepção.
1	paridade	- indicar a paridade do pacote.
72	barramento	- total das linhas

Figura 3 - Tabela do Detalhamento do Barramento Paralelo

ro-tensão. O protocolo de transferência utilizará as linhas *limpo-para-envio, não-autorizo* e *clock-in*. O padrão de transferência da parte útil do pacote poderá ser de 32 bits, com uso pleno do barramento, ou de 8 bits, quando houver necessidade de transmissão mais rápida de pacotes menores, como acontece com pacotes de controle. O uso do barramento no padrão de 8 bits também será necessário quando aparecerem algumas falhas não contornáveis em transmissão com 32 bits. A alteração de padrão de transferência utilizará as linhas *modo-transmissão* e *reformulação*. A realização de testes e ajustamento do subsistema de comunicação será feita através das linhas *reset, teste* e *desconectar*.

Estima-se que o barramento poderá funcionar a uma taxa da ordem de 1 a 10 Mega pacotes/segundo, levando em conta as atividades de requisição do barramento, transferência propriamente dita, e liberação do barramento, em cada transferência. Como cada pacote de informação possui 40 bits (8 para o cabeçalho e 32 para a informação útil), o barramento paralelo proposto corresponderia a um barramento serial funcionando na faixa de 40 a 400 Mega bits/segundo. Desta maneira, pela ampliação do número de linhas de dados, poder-se-ia chegar, com facilidade, num barramento paralelo que trabalhasse a uma taxa correspondente a Giga bits/segundo em um barramento serial.

4. PROJETO DO PROCESSADOR DE COMUNICAÇÃO

4.1. DESCRIÇÃO DO PROCESSADOR DE COMUNICAÇÃO

O processador de comunicação é o elemento responsável pela conexão do usuário (processador operador) à rede. Cabe a ele controlar todo o fluxo de informação entre o processador operador e vice-versa, e, dentre as suas diversas tarefas, destacam-se as seguintes:

- comunicar-se com o processador operador, recebendo e enviando informação;
- manipular os "buffers", colocando e retirando informação;
- controlar o acesso ao barramento;
- estabelecer a comunicação com outro processador de comunicação, cuidando do protocolo de transferência de informação.

O processador de comunicação é formado por 3 módulos: módulo de interface com o barramento; módulo de interface com o processador operador; e módulo de controle das interfaces (figura 4).

O módulo de *interface com o barramento* consiste dos seguintes elementos:

- circuito de acesso ao barramento;
- sincronizadores;
- "buffers" de emissão e de recepção.

O circuito de acesso ao barramento é quem se encarregará da disputa pelo barramento. Ele decidirá, através de prioridade, se o processador de comunicação, no qual está incluido, terá conseguido ou não o direito acessar o barramento naquele instante. A prioridade de cada processador de comunicação será baseada no seu endereço origem dife

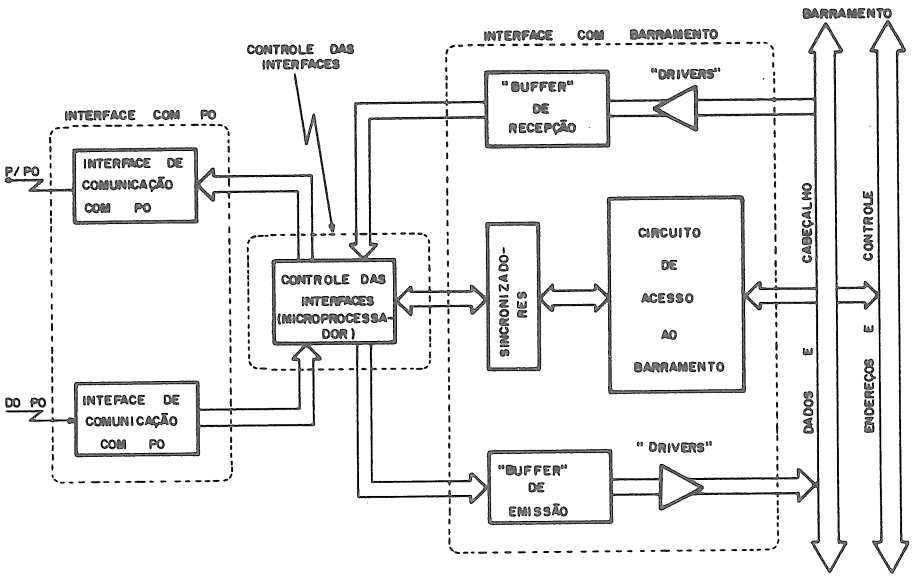


Figura 4 - Configração do Processador de Comunicação

rente de todos os outros, o que garante que a disputa terá sempre um único vencedor.

Os sincronizadores são elementos que retardam ou armazenam sinais necessários à sequencialização das ações de disputa, uso, e liberação do barramento.

Os "buffers" são registradores de dados que colocam ou retiram informação do barramento, em determinados instantes definidos pelos sincronizadores. Os "drives", que estão entre os "buffers" e o barramento, e que também são usados em outras linhas, são circuitos "tri-state", que funcionam como isoladores nesse caso, ou que implementam lógica de fiação em outros casos.

O módulo de *interface com processador operador* consiste de interfaces padrão de entrada e saída de microprocessadores (8251,SIO,etc.) ou de circuitos especiais. A conexão com o processador operador pode assumir muitas variações: as ligações podem ser feitas com o uso de fios ou fibras óticas, de maneira serial ou paralela, com velocidade baixa ou alta, e, quando não se utilizar processador operador, devido a expansão do processador de comunicação, essas linhas poderão estar ligadas a periféricos, sensores, atuadores e outros elementos.

O módulo de *controle das interfaces* consiste de circuitos de microprocessador ou de um hardware especial equivalente. Sua função se resume na coordenação dos trabalhos das interfaces, e na transferência de informação entre elas.

Quando a conexão com processador operador, ou elemento correspondente, exigir

velocidades muito altas, não suportadas por microprocessadores e suas interfaces, os módulos: controle das interfaces, e interface com PO, deverão ser construídos com técnicas e componentes adequados às exigências.

4.2. CIRCUITO DE ACESSO AO BARRAMENTO

O circuito de acesso ao barramento (figura 5), existente em cada processador de comunicação, é o elemento que permitirá que o processador de maior prioridade, dentre aqueles que estiverem concorrendo ao acesso pelo barramento, saia vencedor e ganhe o direito de usar o barramento por um ciclo. Este circuito é um aperfeiçoamento daqueles apresentados em [4] e [9].

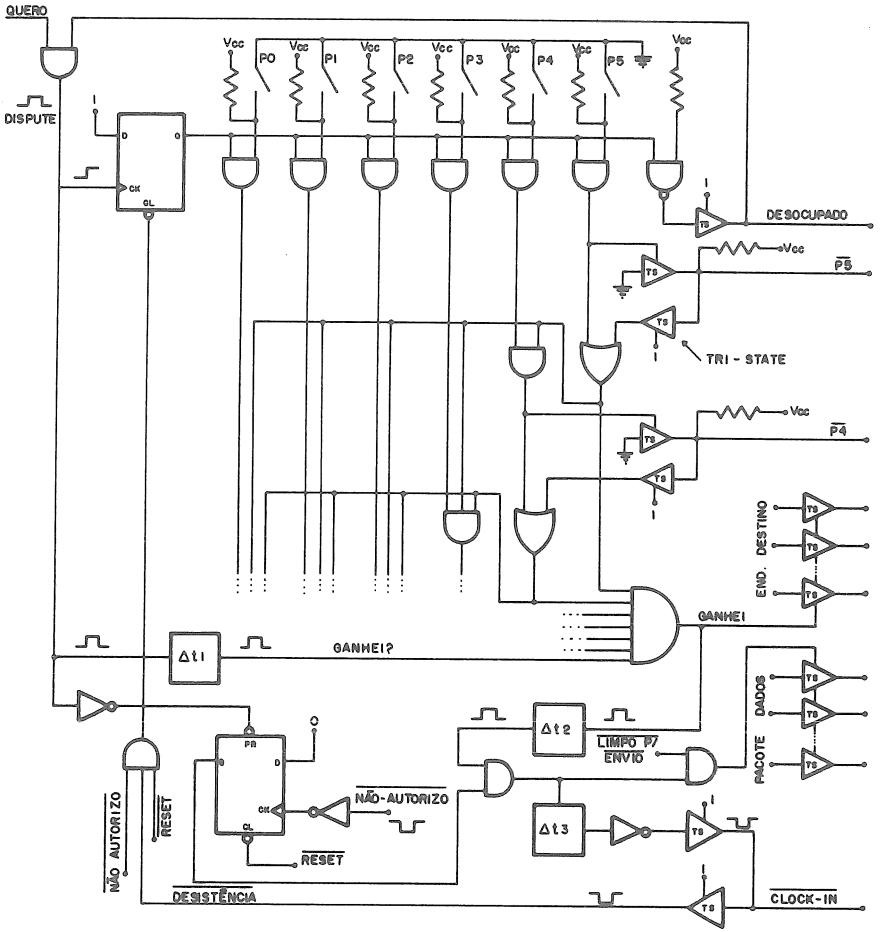


Figura 5 - Circuito de Acesso ao Barramento

O circuito está projetado para 64 processadores de comunicação e funciona da seguinte maneira. Quando um processador de comunicação desejar o domínio do barramento, ele deverá manifestar-se através do sinal *quero*. Se o barramento estiver ocupado (*desocupado = 0*), o pedido ficará pendente até o mesmo desocupar. No caso do barramento estar desocupado, ou ser desocupado (*desocupado = 1*) com pedido pendente, então todos os processadores de comunicação, que estiverem nestas condições, iniciarão simultaneamente uma disputa e inibirão a entrada de outros, indicando que o barramento estará ocupado (*desocupado = 0*). Aqueles, que conseguirem passar, colocarão seus endereços (prioridades) no barramento de endereço origem (arbitragem) e, através da lógica por fiação com componentes "tri-state", será feita a comparação simultânea dos bits de endereço (o maior endereço será prioritário). Os circuitos, que perderem a disputa, deverão esperar a liberação do barramento para participarem da nova disputa (o sinal *quero* de cada um permanecerá). O circuito, que ganhar a disputa, retirará seu sinal *quero* e colocará o endereço destino e o pacote no barramento. O processador de comunicação destino, identificando seu endereço, emitirá o sinal *limpo-para-envio = 0* ou *não-autorizo = 0* se o buffer de recepção estiver livre ou ocupado respectivamente . O sinal *não-autorizo = 0* provocará a liberação do barramento, fazendo *desistência = 0*. O sinal *limpo-para-envio = 0* permitirá a emissão do sinal *clock-in = 0*, que acionará o "buffer" de recepção destino para receber o pacote, e provocará a liberação do barramento para nova disputa.

4.3. EXPANSÃO DO BARRAMENTO E INTERCONEXÃO DE REDES

O problema de expansão do barramento pode ser resolvido com a interconexão de redes através de um elemento denominado conector ("gateway"). A função do conector é receber pacotes de um barramento e colocá-los em outro, com o devido endereçamento. Para isso pode-se adotar diversas estruturas, sendo uma delas a estrutura em árvore (figura 6).

O conector deverá entrar no lugar de um processador de comunicação e consumirá o endereço destino, ao receber um pacote. Para poder continuar sua *viagem*, o pacote deverá ter embutido, em si mesmo, os *passaportes* necessários para que cada conector possa encaminhá-lo adequadamente. Os passaportes serão os endereços dos outros conectores intermediários e do processador de comunicação destino, e deverão, desta forma, reduzir a parte útil do pacote. O fato do pacote, que circular entre barramentos, precisar passar por conectores e ter uma informação útil menor, faz com que a taxa de transmissão da informação sofra uma queda considerável.

5. ASPECTOS DE SOFTWARE

O software de um sistema distribuido envolve várias camadas: o software do processador de comunicação, o núcleo do sistema, abrangendo o subsistema de comunicação e parte dos processadores operadores, o sistema operacional distribuido, e o sistema aplicativo.

O funcionamento do processador de comunicação baseia-se na existência de proce

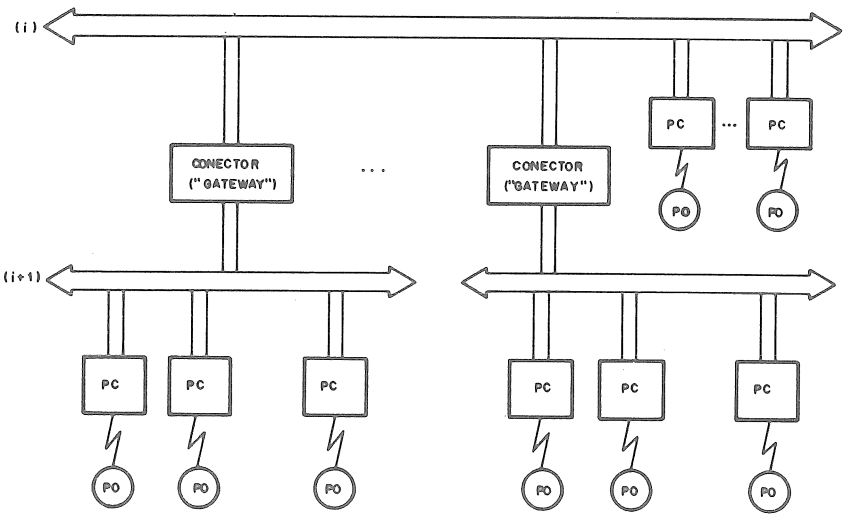


Figura 6 - Uma Forma de Expansão do Barramento (Interconexão de Redes)

dimentos que serão executados no microprocessador do módulo de controle das interfaces. Estes procedimentos é que colocarão o processador de comunicação nas condições iniciais, e depois, farão a monitoração dos sinais de controle e do estado das interfaces, cuidando da transferência interna das informações, e do acionamento de seus circuitos.

A ampliação dos procedimentos do processador de comunicação com primitivas de sincronização e comunicação entre processos e outras primitivas básicas de interesse de sistemas distribuídos, implementará a parcela do núcleo, referente ao processador de comunicação, ficando a parcela complementar para ser implementada no processador operador. O núcleo do sistema distribuído será obtido pela reunião de todos os núcleos de cada par PC-PO [6]. É nessa fase que serão exploradas as questões de implementação de primitivas apropriadas ao controle de aplicações em tempo real, particularmente aquelas relativas à reconfiguração dinâmica de sistema e ao tratamento de exceções.

6. CONCLUSÕES

O projeto do subsistema de comunicação encontra-se detalhado a nível de circuito, e o software mínimo, necessário para o funcionamento do processador de comunicação, já está especificado.

Uma proposta do subsistema de comunicação, com 5 nós de comunicação, está sendo objeto de pedido de financiamento que, se for concedido, permitirá passarmos a fase de implementação e testes.

É interessante notar que, apesar da existência do barramento de alta velocidade, o sistema formado por *"BUFFER" DO PC - BARRAMENTO-"BUFFER" DO PC* constitui-se numa ligação frouxa, que corresponde a uma das características essenciais de sistemas distribuídos.

O barramento centralizado, como foi proposto, permitirá a exploração das vantagens de qualquer barramento, sem os inconvenientes da dispersão física que o tornaria vulnerável, e com a vantagem adicional de ser tolerante à falha em algumas situações.

A possibilidade de utilização de fibra ótica nas ligações PO-PC, e a centralização do subsistema de comunicação, que poderá ficar completamente isolado, tornam este tipo de sistema adequado para aplicações que exijam alta imunidade a ruídos.

O projeto permite muitas variações em todos os seus níveis: ao nível do hardware, pode-se sofisticar o barramento centralizado, o processador de comunicação e as interfaces de conexão com outras redes; ao nível do software básico pode-se conceber vários conjuntos de primitivas, apropriadas a aplicações diferentes, formando vários núcleos; da mesma maneira ao nível do sistema operacional e ao nível da aplicação, o sistema poderá ir de situações bastante simples até as mais complexas.

Um projeto de sistema distribuido, por natureza, é um projeto de fôlego, e esse, particularmente, por abranger a área de aplicações em tempo real, apresenta-nos um desafio a mais. É nossa intenção atuar até ao nível do sistema operacional, mas o projeto estará à disposição dos pesquisadores do Departamento e de fora, interessados em testar variações de hardware e software, e de continuar trabalhando em outros níveis.

7. BIBLIOGRAFIA

[1] ABRAMS, M. et al. - *Computer Networks: A Tutorial*. IEEE Computer Society, N.Y.,1980.

[2] ACAMPORA, A.S. & HLUCHYJ, M.G. - "A New Local Area Networks Architecture Using a Centralized Bus". *IEEE Communications Magazine*, august, 1984, p.12-21.

[3] BRAYER, K. & LAFLEUR, V. - "A Testbed Approach to the Design of a Computer Communications Network". *Computer*, october, 1982, p. 14-23.

[4] CORSO, D.D. & VERRUA, L. - "Contention Delay in Distributed Priority Networks". *The Euromicro Journal - Microprocessing and Microprograming*, january, 1984, p.21-29.

[5] DAVIES, D.W. et al. - *Computer Networks and Their Protocols*. John Wiley & Sons, N.Y., 1981.

[6] KIRNER, C. - "Suporte para Desenvolvimento de Sistemas Distribuidos: Uma Implementação". *Anais do IV Congresso da Sociedade Brasileira de Computação - XI Seminário Integrado de Software e Hardware*, Viçosa, MG, 1984, p. 109-121.

[7] LINDSAY, D.C. - "Local Area Networks: Bus and Ring Vs. Coincident Star". *Computer Communication Review*, july/october, 1982, p. 83-91.

[8] MARQUES, E. - "Projeto de um Barramento Compacto para Sistemas Distribuídos". *Publicação interna*, Departamento de Computação e Estatística - UFSCar, 1984.

[9] TEIXEIRA, C. & TREVELIN, L.C. - "Protocolo de Acesso e Superação de Falhas em Redes de Topologia Estrela Coincidente". *Anais do 2º Simpósio Brasileiro sobre Redes de Computadores, Campina Grande*, PB, 1984, p. 161 - 165.

[10] THURBER, K.J. & FREEMAN, H.A., ed. - *Local Computer Networks: A Tutorial*. IEEE Computer Society, N.Y., 1981.

UMA ARQUITETURA FLEXIVEL PARA SISTEMAS DISTRIBUIDOS TOLERANTES A FALHAS

Autor: Orlando G. Loques F. *

Palavras Chaves: Especificação e Projeto de Sistemas Tolerantes a Falhas, Software Tolerante a Falhas, Arquitetura de Computadores, Sistemas Distribuídos.

Resumo: Este trabalho apresenta a arquitetura de um sistema distribuído tolerante a falhas, o qual suporta redundância dos tipos "hot e cold standby" para tarefas definidas em software. Os dois tipos de redundância são suportados por um conjunto comum de mecanismos os quais tratam da deteção de falhas e da reconfiguração das tarefas da aplicação. Tarefas do tipo cold standby podem ser criadas e ativadas automáticamente pelo sistema de maneira a substituir tarefas em falha, porém informação de estado não é preservada. Para tarefas do tipo hot standby a informação de estado é preservada e uma técnica especial permite que a recuperação de falhas seja obtida de modo transparente. A principal atração desta técnica é permitir que as tarefas do sistema distribuído sejam programadas sem se ter tolerância a falhas em mente; em um estágio posterior do desenvolvimento do sistema as tarefas podem ser transformadas automáticamente de modo a se obter a capacidade de tolerância a falhas.

Abstract: This paper presents the software architecture of a fault-tolerant distributed system providing both cold and hot standby redundancy of software tasks. The two types of redundancy are supported by common mechanisms, which provide for detection of failures and for reconfiguration of the software tasks of the application. Cold standby tasks are created and activated by the system in order to replace failed tasks, but no state information is preserved. Hot standby tasks do preserve state information and provide transparent failure recovery. Transparent recovery is achieved through the use of a special technique that allows the program tasks to be programmed without fault-tolerance in mind; afterwards they can be automatically transformed in order to achieve that capability.

* Engenheiro Eletrônico (PUC/RJ, 1973), Mestre em Engenharia Elétrica (PUC/RJ, 1976), Ph.D. (Imperial College, 1984); arquitetura de sistemas distribuídos, sistemas operacionais, computação tolerante a falhas; Professor do Depto. de Eng. Elétrica da PUC/RJ, Rua Marquês de São Vicente, 225 - CEP 22453 - Rio de Janeiro, Rj.

1. Introdução

Sistemas distribuídos de computadores oferecem varias vantagems em relação a sistemas centralizados. O paralelismo inerente permite melhor desempenho, a capacidade local de processamento permite respostas mais rápidas no tempo, e a modularidade do software e hardware permite que o sistema seja facilmente expandido e modificado a um custo baixo. Uma das maiores vantagens da distribuição e do acoplamento fraco é o potencial para a obtenção da capacidade de tolerância a falhas. Devido ao acoplamento fraco, a falha de componentes não críticos pode levar a operação em modo degradado, ao invés da falha total do sistema. Por outro lado, os componentes críticos podem ser feitos tolerantes a falhas pelo uso de redundância no hardware, a qual devido ao baixo custo dos componentes pode ser suportada nas partes do sistema onde for requerida.

Este trabalho apresenta uma classificação de sistemas em relação aos seus requisitos de confiabilidade. Com base nesta classificação, uma arquitetura "software" para um sistema distribuído de computadores é apresentada. Essa arquitetura é baseada em Conic [6], um ambiente para a programação e configuração de sistemas especialmente projetado para o suporte de sistemas distribuídos embutidos. Contudo, os mesmos princípios são aplicáveis á construção de sistemas tolerantes a falhas baseados em outros ambientes. Os detalhes relevantes de Conic são descritos em um apêndice do trabalho.

2. Tipos de Sistema

A classificação captura a relação entre a falha de qualquer dos módulos que compoêm um sistema e a falha da especificação de confiabilidade associada ao mesmo sistema como um todo. Dois tipos de sistema são identificados:

Baixa Dependência a Falha: O sistema é tal que o padrão de confiabilidade exigido pela aplicação que ele implementa não é violado se um ou mais de seus módulos falha. Isto é, a capacidade de serviço oferecida pelo sistema é de alguma forma degradada devido a falha de um módulo. Contudo, ela ainda é suficiente considerando os requisitos de confiabilidade da aplicação.

Alta Dependência a Falha: O sistema é tal que o padrão de confiabilidade exigido pela aplicacão que ele implementa é violado se qualquer um dos módulos que o constitue falha. Isto é, a falha de um módulo leva a falha do sistema.

Em geral, a classificação apresentada acima simplifica a discussão e a apresentação de sistemas tolerantes a falhas. Embora a maioria dos sistemas possa ser enquadrada no tipo BDF, arquiteturas para o suporte de sistemas do tipo ADF também são necessarias [3,4,11], e alguns aspectos da construcão deste tipo de sistema são pesquisados, eg. [10]. Contudo, deve ser mencionado que sistemas de ambos os tipos podem conviver em uma mesma aplicacão. A arquitetura aqui apresentada pode suportar os dois tipos de sistema.

3. Hipoteses

Para o projeto do sistema foram adotadas as seguintes hipóteses: (i) O projeto é correto, i.e., erros de algoritmo [1] não são tratados, (ii) Confinamento de erros, é assumido que a comunicação entre as tarefas é feita exclusivamente através de mensagens, e que a falha de uma tarefa não causa a falha de outra tarefa através da propagação de erros, i.e, erros não se propagam através de mensagens. Diversas técnicas para a deteçaõ de erros são disponiveis, eg., [2,9]. (iii) Sistema de comunicação confiavel, i.e., ele deve deve ter caracteristicas que não comprometam a confiabilidade do sistema de aplicação: Em particular, é requerido que mensagens em transito não sejam corrompidas e que a rede de comunicação não se particione. O projeto deste tipo de sistema de comunicação é objeto de diversas pesquisas, eg., [5,8].

A hipótese (i) garante que se outros erros não ocorrerem as mensagens serão corretas. Considerando a hipótese (ii) a primeira consequência da falha de uma tarefa será a interrupção do serviço fornecido pelo mesma (falhas transientes podem ser tratadas internamente á estação). Então, considerando (ii) e (iii) falhas de tarefas podem ser detetadas sem ambiguidade pela verificação de parametros de tempo medidos localmente em outras tarefas. Em adição, uma técnica simples (checkpointing) pode ser usada para a recuperação de falhas em sistemas do tipo ADF.

4. O Sistema Tolerante a Falhas

De maneira a fornecer a capacidade de tolerância a falhas foram adicionados ao sistema Conic original dois tipos especiais de tarefas. Ao nivel lógico elas são identicas as tarefas originais de Conic. Contudo, cada tipo especial de tarefa possui uma implentação que lhe adiciona uma forma particular de redundância e supre requisitos diferentes de confiabilidade:

Cold Standby: O qual visa o suporte de sistemas tipo BDF.
Hot Standby: O qual visa o suporte de sistemas tipo ADF.

O projetista especifica cada tipo de tarefa a ser usada na configuração do sistema da aplicação de modo a suprir os requisitos da mesma. Ferramentas padronizadas de desenvolvimento são então usadas para realizar as transformaçoẽs necessárias para a implementação de cada tipo. Mecanismos para suporte durante a operação do sistema também são requeridos. As características de cada tipo são descritas a seguir:

4.1 Cold Standby

Cada tarefa usando este serviço é implementada por uma instância de tarefa padrão em Conic. O projetista especifica um conjunto de estaçoẽs aonde esta instância pode ser alocada. Inicialmente o sistema realiza todas as operaçoẽs de configuração necessárias

para alocar a instância na primeira estação definida no conjunto. Durante a operação do sistema, a estação onde a instância está residindo pode falhar. Após a deteção desta falha, uma nova instância daquela tarefa pode ser criada automáticamente na próxima estação especificada no conjunto associado a tarefa.

Este tipo de tarefa não preserva informação de estado entre duas ativaçoẽs. Um intervalo de tempo razoavel é necessario para que o sistema recrie uma nova instância da tarefa. O sistema assegura a criação da nova instância, contudo, atividades tais como a recuperação de estado e continuidade na cooperação com as outras instâncias de tarefas do sistema devem ser explicitamente programadas pelo projetista.

4.2 Hot Standby

Cada tarefa usando este serviço é implementada por duas instâncias idênticas, e pelo menos duas estaçoẽs são sempre necessárias. O projetista especifica um conjunto de estaçoẽs, o sistema aloca cada instância em uma estação diferente do conjunto. Cada instância está totalmente preparada para execução. Contudo, durante a operação do sistema elas possuem funçoẽs diferentes: Uma delas, chamada ativa, processa mensagens e coopera com as outras instâncias do sistema. A outra, chamada passiva, não executa nenhum processamento, a sua função é manter uma cópia atualizada do estado produzido pela instância ativa. Essa informação de estado é transferida automáticamente para a instância passiva. Inicialmente, o sistema cria e ativa as duas instâncias, assegurando que elas assumam funçoẽs diferentes. Durante a operação do sistema a instância ativa ou passiva pode falhar. No primeiro caso, a instância passiva assume o papel ativo e continua o processamento. No segundo caso, a instância ativa não é afetada pela falha. O sistema pode criar uma nova instância para substituir uma instância em falha; esta instância assume automáticamente o papel passivo.

Este tipo de tarefa preserva informação quando ocorre uma falha da instância ativa e a passiva assume o papel ativo. Considerando que a instância passiva está totalmente preparada, o intervalo de tempo tomado para reparar a tarefa é determinado pelo tempo necessário para detetar a falha e ativar a instância passiva. Está implicitamente assumido que uma falha dupla não ocorre. Depois da falha o sistema pode recriar uma nova instância re-estabelecendo o grau original de confiabilidade. Pode-se suportar mais que uma instância passiva, contudo isto requer mecanismos de suporte mais complicados. Na seção 5 será apresentada uma técnica especial, que independe da aplicação, e permite a recuperação transparente de falhas ao nível da programação e operação das tarefas.

4.3 Mecanismos de Suporte

Em Conic, a constituição do sistema da aplicação é especificada através de uma linguagem de configuração (ver apêndice). De maneira a suportar as tarefas tolerantes a falhas esta linguagem foi extendida de modo a permitir: (1) a especificação do tipo de redundância; e (2) as regras usadas para o controle dinâmico da criação de instâncias. Isto foi alcançado através da declaração abaixo:

```
CREATE  task_name:task_type, parameter_list, redundancy_type
        AT station_set;
```

A mesma estação pode aparecer em conjuntos de estações associados a tarefas diferentes. Neste caso é assumido que esta estação pode suportar somente uma das instâncias separadamente ou tôdas as instâncias ao mesmo tempo. Esta regra é aceitável para o controle de estruturas típicas em aplicações de controle. Contudo, maior flexibilidade pode ser obtida pela introdução de outras regras para o controle da configuração.

Durante a operação do sistema, dois módulos são usados no suporte das tarefas especiais (fig. 1). O Gerente da Configuração (GC) controla a configuração lógica com base em uma tabela gerada a partir da especificação do sistema; suas ações são colocadas em prática através de mensagens enviadas ao sistema operacional de cada estação, o qual suporta a capacidade de reconfiguração dinamica. O GC recebe informação de status gerada pelo Coletor de Status (CS); mudanças de status podem levar a que operações de reconfiguração sejam executadas. De modo a eliminar qualquer ponto singular de falha tanto o GC quanto o CS devem ser feitos tolerantes a falhas. Para o GC isto é conseguido através de sua implementação em hot standby. A obtenção de tolerância a falhas para o CS será discutida na seção 6.

Tarefas do tipo hot standby requerem mecanismos específicos para o suporte da recuperação de falhas, transferencias de estado, e determinação do papel desempenhado por cada instância (ver seção 6). A técnica usada para a recuperação de falhas será apresentada na proxima seção.

5. Técnica para Recuperação de Falhas

A obtenção da capacidade de tolerância a falhas basea-se na adição em Conic de uma primitiva especial de comunicação a qual extende a transação do tipo **pedido-resposta** (ver apêndice). Esta primitiva tem seu uso restrito a tarefas do tipo hot standby e a sua implementação garante que no caso da falha em uma instância, mesmo durante uma transação, esta completar-se-á transparentemente como se a falha não houvesse ocorrido. Isto é, o pedido será executado exatamente uma vez, e somente uma resposta será obtida, consequentemente, o estado da tarefa e do sistema será mantido consistente.

A figura 2 ilustra os fundamentos da técnica. A falha de qualquer uma das tarefas envolvidas pode ocorrer a qualquer momento, particularmente durante uma transação. Considerando que uma instância ativa falha, a instância passiva assume o papel ativo e continua a operação, podendo re-executar parte de seu programa neste processo. Em consequência da re-execução mensagens podem ser re-enviadas (pedido ou resposta); a semântica da transação é mantida pela simples filtragem de mensagens repetidas. Isto é realizado por um mecanismo cujo funcionamento é transparente ao nível das tarefas da aplicação (ver seção 6).

A transferência de estado, necessária para manter a atualidade do estado da instância passiva, e permitir a recuperação de falhas é feita automaticamente por um mecanismo padronizado. Esta capacidade é obtida a partir da especificação das portas que utilizam a primitiva especial de comunicação, as quais são denominadas portas especiais, e das duas regras abaixo:

R1: Uma transferência tem que ser executada após uma mensagem ser consumida por uma porta de entrada especial, e antes que qualquer resultado decorrente do processamento desta mensagem seja enviado para fora da tarefa.

R2: Uma transferência tem que ser executada antes que uma transação seja realizada por uma porta de saida especial, se uma transferência não houver sido executada desde a última transação realizada por esta porta.

Estas regras determinam quando a informação essencial para a sobrevivência do estado da tarefa e recuperação de falhas tem que ser transferida para a instância passiva. O conhecimento da identidade das portas especiais, definidas através da linguagem de configuração, permite que o código padronizado necessário ao cumprimento das regras seja inserido automáticamente nos programas das tarefas por meio de um pre-processador ou compilador. Desta forma, se as comunicaçoes forem restritas ao estilo pedido-resposta, as tarefas de um sistema podem ser programadas sem tolerância a falhas em mente, em uma fase posterior esta capacidade pode então ser obtida.

Uma primitiva especial: save, é disponível para especificar transferência de informação para a instância passiva. Sua implementação garante que a transferência é atómica mesmo em presença de falhas, e que a informação transferida é suficiente para permitir a re-execução da tarefa. Embora não seja requerido, a primitiva save é disponível ao nível da linguagem de programação. Isto permite o tratamento de situacoes particulares a cada aplicação. Como por exemplo: (1) portas especiais e normais podem ser usadas na mesma tarefa; (2) tarefas tipo cold standby e tipo hot standby podem ser interfaceadas.

A figura 3 mostra a aplicação das regras e a utilização da primitiva save. Considere que todas as transaçoes usam portas especiais, neste caso pode ser verificado que em qualquer situação de falha simples, a ação iniciada por req_1 completar-se-á

consistentemente sem a necessidade de qualquer save adicional. Por exemplo, considere que a tarefa_2 falha após executar a primitiva save. Na recuperação, ela executará novamente e gerará req_3, que já poderia ter sido gerada anteriormente na execução prévia. Se req_3 for uma duplicata, já consumida, ela é descartada; o mecanismo de suporte adquire a resposta correspondente: rep_3, e a envia para a tarefa_2. Em caso contrário, req_3 competirá normalmente para ser consumida na interface da tarefa_3. Tambem é possível que antes da falha, a rep_2 tenha sido gerada e consumida pela tarefa_1; na re-execução uma nova instância de rep_2 é gerada e será também filtrada pelo mecanismo de suporte.

6. Detalhes da Implementação

Dois mecanismos são requeridos para o suporte de tarefas em hot standby. O primeiro mecanismo suporta a execução das transaçoes especiais: filtragem de duplicatas e recuperação de mensagens resposta na área de dados da tarefa de entrada. Essas capacidades foram implementadas através de pequenas alteraçoes nas tarefas que suportam intercomunicação no sistema Conic básico. O segundo mecanismo suporta a execução da primitiva save e é implementado por tarefas padronizadas de gerenciamento, as quais são associadas a tarefas em hot standby ao nível da configuração. A invocação da primitiva save é transformada em uma transação pedido-resposta padronizada com a tarefa de gerenciamento, a qual adquire a informação a ser transferida da tarefa da aplicação e a transfere para a instância passiva. As estruturas de dados do núcleo do sistema Conic não precisam ser transferidas; elas são recuperadas pela re-execução da tarefa da aplicação durante a fase de recuperação. Em adição, a tarefa de gerenciamento inicializa o estado de novas instâncias, decide o papel a ser executado pela instância, e deteta falhas da outra instância do par.

No sistema descrito a deteção de falhas é realizada por duas entidades: o (1) Coletor de Status e as (2) Tarefas de Gerenciamento. Em ambos os casos um mecanismo baseado em timeouts o qual requer a geração e o recebimento periodico de mensagens é usado. Cada tarefa de gerenciamento tem a capacidade de detetar a falha da outra tarefa de gerenciamento (e da tarefa da aplicação) associada ao mesmo par. Isto permite a recuperação automatica e independente de falhas de instâncias de tarefas do tipo hot standby. Isto é útil em sistemas pequenos sem um gerente da configuração e para prover tolerância a falhas para o próprio gerente da configuração. De fato, a duplicação da capacidade de deteção a falhas pode ser evitada, contudo isto depende do sistema de comunicação particular. Por exemplo, um sistema de comunicação baseado em passagem de permissão (token) possue inerentemente a capacidade de deteção de falhas localmente a cada estação. Neste caso, deteção de falhas pode ser suprida por um único mecanismo. Correntemente, na PUC/RJ estamos implementando um sistema de comunicação, baseado em passagem de permissão, o qual será integrado ao nosso sistema.

7. Trabalhos Correlacionados

O sistema apresentado em [3] também usa processos duplicados para a obtenção de tolerância a falhas. Contudo, ele não provê uma técnica padronizada para recuperação de falhas, o que tem que ser feito caso a caso pelo programador. Em adição, a interface de entrada dos processos é definida por uma única fila de mensagens, o que não permite uma escolha não determinística das entradas nem o uso de guardas, como é possivel em Conic. Isto pode tornar a programação de algumas aplicaçoes diffcil.

Em [10] uma outra técnica para a construção de programas distribuídos tolerantes a falhas foi proposta. Ela requer a definição explícita de açoes tolerantes a falhas ao nível do programa, o que requer a intervenção cuidadosa do programador. Em adição, as comunicaçoes são realizadas através de variaveis compartilhadas. Isso não permite que as tarefas sejam projetadas independentemente, e é um conceito menos natural em um sistema distribuído onde mensagens tem que ser usadas em algum nível da implementação. Desta forma, o uso da técnica apresentada neste trabalho pode ser considerado vantajoso.

8. Conclusoes

Pela especificação do tipo de redundância requerido a capacidade de tolerancia a falhas para as tarefas é obtida automaticamente. A arquitetura é bastante simples o que contribue para que erros de projeto sejam menos prováveis. Os mecanismos de suporte são implementados por módulos escritos em Conic básico, o que facilita o seu transporte para qualquer tipo de processador já suportando essa arquitetura. Por outro lado os mecanismos específicos para o suporte de sistemas do tipo ADF são ortogonais aos mecanismos que suportam a capacidade de reconfiguração dinâmica (CS e GC), os quais são suficientes para o suporte de sistemas tipo BDF. Desta forma eles podem ser independentemente modificados e seletivamente usados para suprir as necessidades de cada aplicação particular. Finalmente deve ser ressaltado que, em princípio, para tarefas em hot standby, a tolerância a falhas pode ser obtida de modo transparente. Contudo, o sistema é flexível de modo a poder se adaptar a situaçoes específicas. Uma apresentação mais completa do sistema descrito neste trabalho, bem como dos detalhes de sua implementação pode ser encontrada em [7].

9. Apéndice

A linguagem de programaçcão Conic é baseada em Pascal, a qual foi extendida de maneira a prover interfaceamento através de portas e primitivas para troca de mensagens. O componente basico é o módulo tarefa (processo) que é equivalente a um programa Pascal sequencial, e pode ser separadamente projetado e compilado. A interface de uma tarefa é definida por um número de portas de entrada e saida. As primitivas de comunicação suportam dois tipos de transação: O tipo **pedido-resposta** (fig. 4), é projetado para uso

quando uma resposta, ou confirmação da aceitação, do pedido for requerida. Depois de enviar a mensagem, a tarefa fonte, tem a sua execução suspensa, sendo somente ativada após o recebimento da resposta. O tipo **notificação**, não prove qualquer segurança sobre o destino da mensagem de saida, a tarefa fonte continua executando após o envio da mensagem; esta transação permite flexibilidade.

A linguagem de configuração de Conic é usada para especificar um sistema (fig. 5) através da seleção e interconexão de várias instâncias de módulos. A declaração **USE** identifica o conjunto de tipos de módulos a ser usado na construção do sistema; a declaração **CREATE** especifica os nomes das instâncias a serem criadas no sistema; a declaração **LINK** especifica a interconexão das instâncias através da ligação de portas de saida com portas de entrada.

10. Referências

[1] Anderson, W., Lee, P., Fault Tolerance Principles and Practice, Prentice Hall International, 81.

[2] Andrews, D., Using Executable Assertions for Testing and Fault-Tolerance, 9th Symposium on Fault-Tolerant Computing, Junho 79.

[3] Bartlet, J., A Non-Stop Kernel, Proc. of the Eighth Symposium on Operating Systems Principles, Dezembro 81.

[4] Borg, A., et al, A Message System Supporting Fault Tolerance, Proceedings of the Ninth Symposium on Operating Systems Principles, Outubro 83.

[5] Kleinrock, K., et al, A Highly Reliable Distributed Loop Architecture, 10th Symposium on Fault-Tolerant Computing, Outubro 80.

[6] Kramer, J., et al., Conic: An Integrated Approach to Distributed Control Systems, IEE Proc., Vol. 130,Pt. E.N. 1, Janeiro 83.

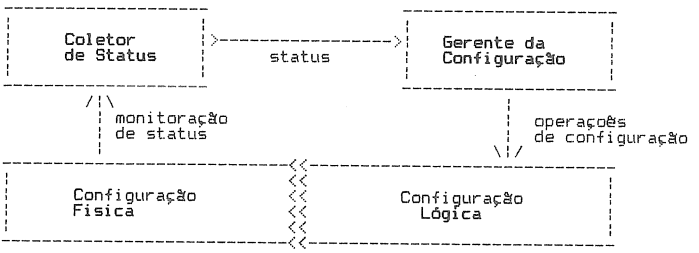
[7] Loques, O., G., A Fault Tolerant Distributed Computer Control System, Tese Ph.D, Imperial College, Londres, Fevereiro 84.

[8] Powell, R., Dependability Evaluation of Communication Support Systems for Local Area Distributed Computing, 12th Symposium on Fault-Tolerant Computing, Junho 82.

[9] Rennels, D., et al, A Study of Standard Building Blocks for the Design of Fault-Tolerant Distributed Computing Control Systems, 8th Symposium on Fault-Tolerant Computing, Junho 78.

[10] Scheneider, F. B., Schlichting, R., Towards Fault-Tolerant Process Control Software, 11th Symposium on Fault-Tolerant Computing, Junho 79.

[11] Wensley, H., et al., The Design and Analysis of a Fault Tolerant Computer for Aircraft Control, Proceedings of the IEEE, Outubro 78.



<< : Mapeamento Tarefas-->Estaçoës

Fig. 1 Funcionamento dos Mecanismos de Suporte

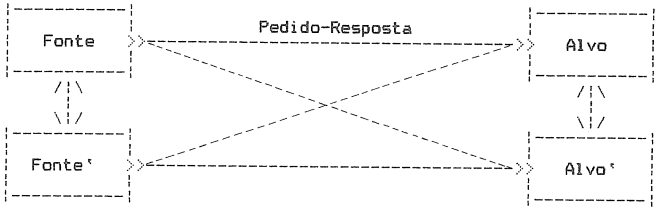


Fig. 2 Execução da Transação Especial

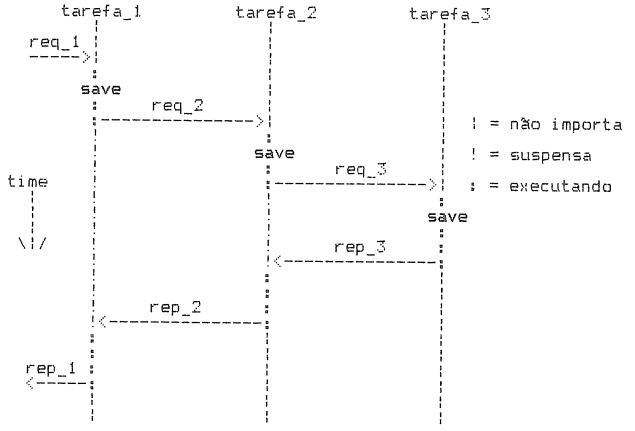


Fig. 3 Regras para uso da Save

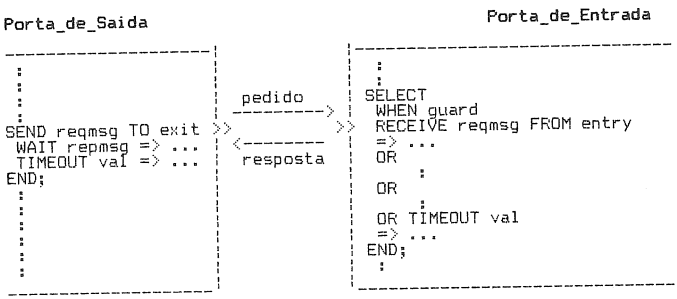


Fig. 4 Primitiva Pedido-Resposta

```
SYSTEM    patient_monitoring;
USE       bed_monitor, nurse_unit;
CONST     nbed = 4;
CREATE    bed[1..nbed]: bed_monitor;
          nurse: nurse_unit;
LINK      bed[1..nbed].alarms  TO  nurse.alarms;
          nurse.query[1..nbed]  TO  bed[1..nbed].status;
END;
```

Fig. 5 Exemplo de Especificação de Sistema

P. KEGLEVICH DE BUZIN *

SUMARIO

Este trabalho aborda o problema de concepção de sistemas distribuídos. São estabelecidos alguns conceitos fundamentais, e é proposto um modelo para descrever um ambiente computacional distribuido. A partir deste modelo é constatado que a descrição de sistemas distribuídos introduz um novo estilo de programação.

ABSTRACT

This paper deals with the problem of distributed systems conception. It establish some fundamental concepts and proposes e model to describe an distributed computational environment. From this model it was found out that the description of distributed systems introduces a new stile of programation.

* MsC. Ciência da Computação (UFRGS, 1984); Sistemas Distribuídos, controle de processos e sistemas em tempo-real, engenharia de software, projeto de linguagens de programação. UFRGS-PGCC, Av. Osvaldo Aranha, 99 - Porto Alegre, RS - CEP 90000.

1. INTRODUÇÃO

Avanços tecnológicos tem tornado viável construir sistemas complexos sobre uma rede de computadores. Para assistir tais sistemas existe uma crescente necessidade por maneiras efetivas de organizar e manter sistemas distribuídos. Assim os custos do equipamento reduzem-se assombrosamente, porém surge um impasse, que um número ainda insuficiente de pesquisadores está tentando resolver: a complexidade e o custo do desenvolvimento do software |MCQ 80|.

Neste trabalho procura-se obter uma abordagem adequada para a concepção de sistemas distribuídos, para tanto é necessário primeiro um entendimento apropriado de tais sistemas, e a seguir é feita uma proposta de modelo para ambientes computacionais distribuídos, que irá fornecer uma abordagem para um projeto de sistemas distribuídos.

2. O PROBLEMA DA CONCEITUAÇÃO

Para o desenvolvimento de sistemas distribuídos é fundamental um respaldo de um bom entendimento de conceitos consistentes sobre o tipo de sistema a ser desenvolvido. No caso de sistema distribuídos, este problema é fortemente sentido. A carência de um entendimento mais acurado nesta área torna a tarefa de projeto de uma aplicação distribuída tremendamente complexa e onerosa. Como exemplo, cabe observar que a própria definição do que seja um sistema distribuído é obscura, pois tenta-se definí-lo em função da aplicação, o que torna difícil a obtenção dos conceitos fundamentais para estruturar um sistema de computação distribuída.

3. CARACTERIZAÇÃO DOS SISTEMAS DISTRIBUÍDOS

De uma maneira informal, um sistema distribuído pode ser caracterizado como sendo um conjunto de elementos processadores autônomos que se comunicam para realizar um sistema coerente de computação. Cada elemento processador possui pelo menos um processador operante, além de memória local. A operação de cada elemento é assíncrono em relação aos outros. A comunicação é efetuada através de uma rede de interconexão, cuja topologia pode ser qualquer, embora usualmente seja em barra ou em anel |FRA 81|, como no caso de redes locais de computadores.

3.1 POTENCIALIDADES

O desenvolvimento dos sistema distribuídos é motivado por vários benefícios antevistos, além de outros ainda a serem constatados. A importância de tais benefícios, e a habilidade para materializá-los depende do ambiente da aplicação. Alguns destes benefícios são citados a seguir:

a) melhor confiabilidade e recuperabilidade em relação aos sistemas centralizados,
b) Extensibilidade em tempo de carga ou execução,

c) melhores condições de aumentar o desempenho que os sistemas centralizados,
d) a melhora de desempenho tem o potencial de poder ser feita a um custo menor que em sistemas centralizados.
e) permite uma distribuição da capacidade de processamento, reduzindo com isto o custo de transmissão de dados,
f) finalmente, possui o potencial de ser mais fácil de programar que sistemas centralizados.

O potencial de ser mais fácil de programar ainda não foi totalmente constatado na prática, em parte devido ao rústico entendimento atual das relaçãoes entre os requisitos do processamento de aplicações e os necessários mecanismos de comunicação entre processos |FRA 81|.

3.2 SISTEMAS ABERTOS X SISTEMAS FECHADOS

Basicamente os sistemas distribuídos podem ser classificados em duas grandes classes: os sistemas abertos (OSI) e os sistemas fechados. Para os sistemas abertos já existe uma recomendação largamente aceita para a estruturação dos níveis de protocolo, que é a recomendação ISO. Porém, embora esta recomendação se adapte bem para grandes redes de teleprocessamento, a mesma já apresenta problemas para ser usada em redes locais. As redes locais não precisam de uma estrutura de protocolo tão complexa quanto a dos sistemas de transmissão de dados a longa distância, onde é comum um grande número de nodos com sistemas heterogêneos interconectando-se e desconectando-se dinamicamente.

Para os sistemas fechados não existe uma recomendação consagrada como o caso da ISO |DAT 81|, mas existe certo consenso quanto ao tipo de arquitetura |ADV 80|. A abordagem conceitual neste caso é totalmente diferente do caso dos sistemas abertos. No caso dos sistemas fechados toda a rede é considerada como a arquitetura de uma máquina, ou seja, a máquina alvo é a rede. No caso dos sistemas abertos, o nodo é a máquina alvo, e a rede é encarada como um recurso disponível a esta máquina. Todo o padrão ISO baseia-se neste conceito.

Se for comparada a estrutura de níveis do modelo de sistema fechado com a estrutura do modelo de referência da ISO, será observado que o modelo da ISO enquadraria os níveis de aplicação, sistema operacional e kernel distribuído do sistema fechado como somente o nível de protocolo de aplicação. Por outro lado, devido a sua característica, o modelo do sistema fechado não se preocupa com os níveis de apresentação e sessão que existem no modelo de referência da ISO. Portanto o modelo de sistemas abertos são totalmente diferentes do modelo dos sistemas fechados, e não podem ser confundidos. Assim para o caso de redes locais, onde se necessita de uma estrutura de protocolos mais simples, o modelo de sistemas fechados se presta muito mais do que o modelo OSI.

4. A OBTENÇÃO DE UM MODELO PARA SISTEMAS DISTRIBUIDOS

Para a obtenção do modelo deste ambiente computacional, é necessário fazer uma consideração mais apurada no que concerce a interação entre processos, a identificação de elementos dentro do ambiente, o tipo de controle, a proteção de recursos e a sincronização de processos. Tais aspectos são considerados na abordagem do problema dos sistemas distribuídos, e a seguir é apresentado um modelo que descreva o ambiente computacional distribuído. Maiores detalhes sobre estas considerações podem ser encontrados em |KEG 84a|.

4.1 O PROBLEMA DOS SISTEMAS DISTRIBUÍDOS

A peculiaridade dos sistemas distribuídos é que os mesmos são sistemas fracamente concetados, devido a arquitetura do hardware, para os quais o estado global não é conhecido. Esta característica causa o surgimento de vários problemas para o projeto do software. A solução destes problemas leva a um entendimento mais acurado do que seja um ambiente de programação distribuído e consequentemente permite definir com mais clareza os problemas das aplicações distribuídas, assim como os problemas das aplicações em qualquer ambiente computacional.

Os principais problemas que surgem, devido as características dos sistemas distribuídos, são a maior dificuldade de controlar a ocorrência de eventos (sincronização), que se reflete no problema da manutenção de consistência do sistema, e a diversidade de aplicações e operações que podem ser feitas ao mesmo tempo sobre tal sistema. A diversidade de aplicações ao mesmo tempo irá se refletir na abordagem do modelo de controle de execução do sistema distribuído, e tornará o problema de proteção mais complexo que em sistemas operacionais.

É importante observar que é o modelo de controle de execução que vai dar a característica de sistema distribuído a um determinado ambiente computacional. Assim, por exemplo, um sistema com vários nodos, mas com um nodo mestre que controla todos os outros, não necessariamente pode ser caracterizado como sistema distribuído, embora tenha nodos distribuídos geograficamente. Tal sistema deve antes ser caracterizado como algo similar ao convencional sistema de teleprocessamento |ADV 80|. A maior complexidade dos sistemas distribuídos surge exatamente do fato de se constituir em vários processadores autônomos que devem cooperar juntos para produzir uma computação coerente. Portanto um sistema distribuído é principalmente caracterizado por um modelo de controle de execução distribuído, ao passo que um sistema centralizado é caracterizado por um modelo de controle de execução centralizado.

A abordagem do problema da sincronização implica em uma análise mais acurada do controle da ocorrência de eventos em sistemas distribuídos, examinando a nominação e endereçamento em sistemas computacionais, os mecanismos de sincronização e os tipos de interção entre processos. É interessante notar que existem basicamente três modelos de interação entre processos |AND 81|: monitor (ou procedure), passagem de mensagem e operações remotas. Tais modelos permitem analizar as classes de mecanismos de sincronização mais adequados para cada tipo de ambiente computacional e para cada tipo de aplicação |AND 83|.

4.2 ENTENDENDO UM AMBIENTE COMPUTACIONAL DISTRIBUÍDO

Um ambiente computacional pode ser visto como um conjunto de recursos disponíveis dentro de um mesmo sistema e acessíveis de uma maneira uniforme, de acordo com as regras do ambiente.

4.2.1 TIPOS DE AMBIENTES

Foram considerados quatro tipos diferentes de ambientes computacionais: distribuído, paralelo, concorrente e sequencial. Devido as suas características, estes ambientes relacionam-se fortemente, de forma que um ambiente distribuído engloba um ambiente parelelo, que por sua vez engloba um ambiente concorrente, que por sua vez angloba um ambiente seqüencial.

Assim, ate ao nível de ambiente concorrente, inclusive, não existe preocupação com a localização de unidades de programa. Com o processamento paralelo já existe a preocupação com a localização das unidades de programa ou com a associação algoritmo-processador para a computação de um programa, sem no entanto preocupar-se com a característica do meio de comunicação. Já em um ambiente distribuído existe a preocupação com o meio de comunicação, pois é preciso controlar as situações de erro e de retardo de mensagens.

Quanto ao tipo de recursos que oferecem, os ambientes computacionais podem ser subdivididos em ambientes físicos (hardware) e ambientes lógicos (software ou parte algoritmica). Portanto, para formar um ambiente computacional é necessário localizar sobre o ambiente físico o ambiente lógico.

4.2.2 AMBIENTES FÍSICOS

O ambiente físico pode ser somente uma máquina hospedeira, para o caso de ambiente computacional concorrente, ou uma arquitetura com vários processadores, para o caso de um ambiente paralelo, ou uma rede, para o caso distribuído. Assim, os elementos de um ambiente físico podem ser processadores, memórias compartilhadas ou nodos de uma rede.

No caso de ambientes físicos de sistemas distribuídos, deve ser considerado um aspecto importante que é a característica do meio de comunicação. De um modo geral, um sistema físico pode ser considerado como fracamente concectado quando possui retardo, de modo que a produção de um evento e a sua materialização é distinguível. Isto sem dúvida torna bastante complexo o problema da sincronização e controle de eventos. Por outro lado, o sistema pode ser assumido como fortemente conectado quando não há retardo na comunicação, ou seja, a produção de um evento e a sua materialização são indistinguíveis. Como exemplo do último caso tem-se uma arquitetura de multiprocessadores que compartilham uma memória.

4.2.3 AMBIENTES LÓGICOS

A estrutura da parte lógica de um ambiente computacional deve ser totalmente modular para poder comportar aplicações complexas. Assim o problema da manutenção de consistência de um sistema, que reside no controle da ocorrência dos eventos, fica subdivídido em problemas menos complexos. Dentro do ambiente, tais módulos só se comunicam através de mensagens.

Estas mensagens dentro do ambiente lógico podem ser de tipos diferentes conforme a semantica associada no envio e recebimento das mesmas. Assim estas mensagens podem ser operações, exceções em execução, ou interrupções. As operações podem implementar desde o sistema de mensagens comum, até as chamadas remotas de procedure |AND 81|. Desta forma, dentro deste modelo é feito um tratamento hogeneo tanto para exceções como para a sincronização de processos.

4.2.4 O INTERRELACIONAMENTO ENTRE AMBIENTES

No caso do interrelacionamento entre dois ambientes computacionais é usado o conceito de interface computacional. Com isto o modelo permite a convivência de vários ambientes computacionais sobre o mesmo sistema, facilitando a implementação de um sistema distribuído com vários aplicativos diferentes. Por outro lado, no caso de relacionamento entre ambiente lógico e componente de ambiente físico, é usado o conceito de interface hardware. O relacionamento global entre todo o ambiente físico e todo o ambiente lógico é estabelecido na construção do ambiente computacional.

5. CONCLUSÕES

O presente trabalho constitui em um estudo preliminar que precedeu o projeto de uma ferramenta de suporte para sistemas distribuídos |KEG 84a, KEG 84b|. A abordagem do projeto é fornecer uma ferramenta básica onde os problemas básicosinerentes a sistemas distribuídos

já estejam resolvidos, de modo que a preocupação com sincronização e roteamento de mensagens assim como com o fornecimento de outros recursos, fica aliviada da implementação de uma aplicação em um sistema distribuído.

Uma das principais conseqüências desta modelagem preliminar é a introdução de recursos, na nova linguagem projetada, de descrever o meio físico (ou arquitetura de hardware) e a localização de módulos sobre esta arquitetura. A maioria das linguagens para sistemas distribuídos desenvolvidas atualmente não se preocupam com este detalhe. Isto é feito externamente, o que torna o sistema ainda mais complexo, pois não há um tratamento uniforme neste caso. Assim, não é uma tarefa trivial mudar uma aplicação de uma rede para outra, por exemplo.

O modelo de ambiente computacional fornece ao programador de sistemas dois pontos de vista do ambiente de programação: o ponto de vista físico e o ponto de vista lógico. Portanto o programador de um sistema distribuído deve ter condições de:

a) Especificar a distribuição das unidades do programa.
b) Estabelecer processos, programando as partes que podem ser executadas em paralelo.
c) Desenvolver os mecanismos de comunicação entre os componentes do programa, projetando a política de sincronização entre os processos.
d) Providenciar estratégias para o manuseio de exceções.

Estas características introduzem um novo estilo de programação, que é diferente dos já existentes, correspondentes a programação seqüencial e concorrente |TOB 82|.

BIBLIOGRAFIA

|ADV 80| ADVANCED COURSE ON DISTRIBUTED SYSTEMS ARCHITECTURE AND IMPLEMENTATIONS, Munchen Mar. 4-13, 1980. Distributed systems architecture and implementation. Berlin, Springer-Verlag, 1981. (Lecture Notes in Computer Science, 105)

|AND 81| ANDREWS, G.R. Synchronizing resources. ACM Trans. on Programming Languages and systems, 3(4):405-30, Oct. 1981.

|AND 83| _____. & SCHNEIDER, F.B. Concepts and notations for concurrent programming. Computing Surveys, 15(1):3-43, Mar.1983.

|DAT 81| DATA processing - Open systems interconnection - Basic reference model. Computer Networks, 5(2):81-118, Apr.1981.

|FRA 81| FRANTA, W.R. et alii. Real time distributed computer systems. In: ADVANCES in Computer. New York, Academic Press, 1981. v. 20, p.40-82.

|KEG 84a| KEGLEVICH DE BUZIN, P.F.W. Uma abordagem para a concepção e implementação de sistemas distribuídos. Porto Alegre, PGCC da UFRGS, 1984. (Dissertação de mestrado).

|KEG 84b| _____. Constructor: projeto de uma ferramenta para descrição de computadores. Prêmio SERPRO de monografia âmbito nacional, primeiro lugar. Nov. 1984.

|MCQ 80| McQUILLAN, J.M. Local network technology and lessons of history. Computer Networks, 4(5):235-8, Oct. 1980.

|TOB 82| TOBIASCH, R. & RAFFLER. H. Configuratig software - a method for bridging over the gap between concurrent processing and distributing processing. In: INFOCOM 82, Nevada, Mar/Apr. 1982. Proceedings. s.n.t. p. 44.8

José A.M. QUEIROZ*

SUMÁRIO

Este trabalho apresenta o kernel distribuído para suporte da linguagem Constructor. Nele são descritas as funções do kernel, as estruturas de dados e as primitivas do kernel. Considerações são feitas sobre uma implementação do ambiente Constructor na UFPE.

ABSTRACT

This work describes the distributed kernel Constructor. It also gives the kernel functions, the data structures, and the kernel primitives. An intended implementation of the Constructor environment is presented.

* Engenheiro Civil (EE/UFPE - 1973) e Mestre em Ciência da Computação (PGCC/UFRGS-1984); Sistemas Distribuídos e Sistemas Operacionais Distribuídos; Professor Assistente do Departamento de Informática da UFPE, Analista de Sistemas do Núcleo de Processamento de Dados da UFPE e membro do Grupo de Redes de Computadores e Sistemas Distribuídos da UFPE; REDIS/UFPE - Caixa Postal 7474 - Recife - PE; Fone: (081) 271 2510; TELEX (081) 1267.

O KERNEL DISTRIBUÍDO CONSTRUCTOR

INTRODUÇÃO

A construção de um sistema distribuído envolve o projeto de diversos elementos que, em seus funcionamentos, trabalham cooperativamente para que recursos dispersos sejam utilizados na prestação de diversos serviços. O modelo de arquitetura de sistema distribuído (QUE 84) se baseia no modelo de objeto (JON 78) onde a interação entre processos é feita através da invocação de operações definidas por outros processos (AND 82).

A base para implementação do modelo de sistema distribuído é constituída por um conjunto de sistemas de computação autônomos, interconectados através de um subsistema de comunicação. Além do nível correspondente ao subsistema de comunicação, o modelo envolve três níveis: kernel, sistema operacional e aplicações. O kernel de cada nodo oferece, entre outras coisas, facilidades para sincronização e intercomunicação de processos e facilidades para atendimento de interrupções. Além disso, ele trata da comunicação com o subsistema de comunicação. O kernel distribuído é constituído pelo consórcio de todos os kernels dos nodos componentes interconectados através do subsistema de comunicação. Acima dele, o restante do sistema distribuído (nível de sistema operacional e nível de aplicações) é implantado por processos que interagem através das funções implementadas pelo kernel.

Cada novo componente possui uma cópia do kernel que fornece os mecanismos necessários para que os conceitos de operações e objetos sejam implementados. A interação entre processos pode ocorrer para processos em um mesmo nodo (interação local) ou em nodos distintos (interação remota). O kernel deve fornecer mecanismos para que processos interajam do mesmo modo, independentemente da localização dos mesmos e independentemente de serem processos fornecidos pelo sistema ou por usuário.

O kernel componente é constituído de um conjunto de rotinas e estruturas de dados que implementam as funções básicas (primitivas) do sistema executadas de forma não interrompível. Os processos se comunicam com o kernel através de instruções especiais ('traps'), tipo chamadas do supervisor, e, através delas, solicitam serviços. O processador é passado ao kernel após um 'trap'. Os dispositivos e o relógio pedem a atenção do kernel através de interrupções. Quando o kernel não está em execução e uma interrupção ocorre, o processador é passado para ele. Durante a maior parte do seu tempo o kernel deverá estar envolvido na execução de facilidades para interação de processos. Para interações remotas ele se utilizará dos serviços do subsistema de comunicação.

A definição do kernel busca atingir metas que o tornem tão pequeno e simples quanto possível. Esta minimização de tamanho e de complexidade aumenta a probabilidade de se obter uma implementação correta para ele. Os kernels pequenos, simples e também práticos oferecem as condições de segurança para que o software que vai ser suportado possa ser mais confiável (POP 78). O kernel deve comportar todo o software que se relacione com a segu-

ça do sistema. Funções não importantes para a segurança do sistema podem ser implementadas por processos em um nível acima do kernel.

FUNÇÕES DO KERNEL

A tarefa principal a ser realizada pelo kernel é a de implementar os conceitos de programação da linguagem Constructor (BUZ 84). Com isto um ambiente de programação Constructor poderá ser fornecido aos usuários, permitindo a definição de processos para serem executados nos diversos componentes do sistema distribuído. Cada processo terá para si sua própria unidade de processamento virtual. Estas unidades virtuais são como as unidades reais; a diferença entre elas é apenas a menor velocidade de execução das unidades virtuais que são implementadas pela multiplexação das unidades reais, o que resulta numa taxa de execução variável para os processos.

As funções que o kernel executa para implementar um ambiente Constructor e para atender às necessidades dos processos, dando a cada um a impressão de possuir sua própria unidade de processamento, se dividem entre: funções para implementação e gerenciamento de processos, funções para sincronização e intercomunicação de processos, funções para gerenciamento de áreas de armazenamento para passagem de parâmetros, e funções para manipulação de interrupções e execução de operações de entrada/saída.

Implementação e Gerenciamento de Processos

Os processos são entidades de execução independentes que podem interagir com outros processos através de operações. Os processos executam, ao menos conceitualmente, de forma concorrente e têm acesso exclusivo às suas variáveis locais.

Cada processo é representado no kernel por um registro descritor. O registro descritor contém informações necessárias para colocar o processo em execução; estas informações incluem o contador de instruções, os registradores gerais e de endereçamento, e outras. Isto permite que o processador seja multiplexado pelos diversos processos que concorrem por ele.

Ao serem inicializados os processos são colocados em uma fila de processos aptos ('ready') para receberem o processador. Sempre que acaba uma tarefa, o kernel deve selecionar um processo na fila de aptos para receber o processador. O escalonamento faz com que um processo seja retirado da fila de aptos e que seu ambiente de execução, que estava guardado no seu descritor, seja restaurado. A fila de aptos é organizada de acordo com a prioridade e com o tempo de entrada dos processos. O processo escalonado assume o controle do processador e fica em execução ('running') até que ele necessite de um serviço do kernel ou seja interrompido. A solicitação, por um processo, de serviços do kernel faz com que o estado atual do solicitante seja salvo e que ele seja bloqueado ('blocked') a espera do atendimento do serviço solicitado. Quando o serviço solicitado for concluído o processo que estava bloqueado passará para a fila de aptos. Quando um processo em execução é

interrompido, ou pelo fim de sua fatia de tempo ou por uma interrupção de dispositivo, o estado atual do processo é salvo e ele é colocado na fila de aptos. Um processo em execução pode decidir por seu próprio encerramento e isto fará com que seu descritor seja marcado como correspondente a processo terminado. O término normal de um programa Constructor ocorrerá quando terminarem todos os processos. Mecanismos para que processos exerçam controle sobre outros processos fazem com que novos estados sejam incorporados aos que foram descritos.

Sincronização e Comunicação entre Processos

Embora independentes, os processos devem interagir para executar as suas tarefas, as quais, em geral, são partes (sub-tarefas) de uma tarefa maior, comum a todos, a ser executada de modo cooperativo.

Os processos se comunicam e sincronizam através de operações. Um processo que deseja interagir com outro pode fazê-lo através da invocação de uma operação que este outro implemente. Através dos parâmetros da invocação o processo invocante envia informações para o processo invocado. Esta invocação pode ser uma invocação síncrona feita através de um comando de chamada CALL, ou uma invocação assíncrona feita através de um comando de envio SEND. A operação invocada estará implementada em um processo, embutida em um comando de seleção SELECT. Este comando, cada vez que for executado, selecionará uma operação para atender, dentre um conjunto de operações pendentes. Se a invocação atendida tiver sido feita por um CALL, uma resposta será enviada para o processo invocante. O processo invocado pode passar informações para o processo invocante através dos parâmetros de saída da invocação. Complementando este mecanismo para sincronização e comunicação entre processos, uma facilidade é fornecida para que diversas invocações síncronas (CALLS) possam ser disparadas concorrentemente, o comando de invocação concorrente CONC. O comando CONC permite que um processo possa invocar concorrentemente diversas operações que podem estar implementadas por diversos processos. O processo invocante será bloqueado depois que todas as invocações forem feitas e, voltará a ficar apto e concorrer pelo processador depois que todas as respostas às invocações feitas sejam recebidas.

Os processos que interagem podem estar localizados em nodos processadores diferentes. Para um processo a interação com um outro é feita de forma independente da localização desse outro. Quem distingue entre invocação local e invocação remota é o kernel, o qual, no caso de interações remotas lança mão do subsistema de comunicação.

A cada operação implementada está associada uma lista com invocações pendentes para a operação. Uma invocação pendente indica qual o processo que fez a invocação e onde estão os parâmetros atuais da invocação. Estas listas são mantidas pelo kernel.

A invocação de uma operação faz com que o kernel insira um novo elemento na lista de invocações pendentes para a operação, contendo: identificação do processo, tipo de invocação e ponteiro para área de parâmetros. Após, é verificado se o processo que implementa

a operação invocada está bloqueado e se a invocação corrente pode fazer com que esse processo seja continuado. Se isto ocorrer o kernel acorda o processo, colocando-o na fila de aptos. Se a invocação foi feita por um comando CALL, o processo invocante é bloqueado e um novo processo é escalonado. Se a invocação foi feita por comando SEND, o processo invocante não é bloqueado e continua executando. A invocação feita por comando CALL dentro do comando CONC origina ações que são uma mistura das ações correspondentes aos comandos SEND e CALL. O processo que fez invocações concorrentes só será bloqueado depois que todas as suas chamadas CALL forem disparadas. Para este processo ser colocado novamente na fila de aptos é necessário que ele receba todas as respostas das operações que foram invocadas.

Para que um processo atenda a uma invocação de operação é necessário que o seu comando de seleção SELECT que implementa a operação seja executado. A execução do comando SELECT está dividida em duas partes: na primeira delas é feita a seleção de uma operação para executar e na segunda os comandos que implementam a operação selecionada são executados, de acordo com os parâmetros da invocação escolhida para ser atendida. Durante a fase de seleção uma operação pode ou não ser escolhida para execução. Se não há seleção de uma operação e o comando possui a cláusula ELSE, os comandos dessa cláusula são executados e a execução do SELECT é concluída. Caso uma operação não tenha sido selecionada e o comando SELECT não possua cláusula ELSE, este processo deve ser bloqueado. Quando o bloqueio do processo é feito são passados como argumento desse bloqueio uma máscara que indica as operações que, se invocadas, podem fazer com que o comando seja completado, e uma informação que indica se a alteração de variáveis globais do recurso podem tornar verdadeiras uma ou mais expressões de sincronização. A execução do comando SELECT é retardada até que uma operação que ele implemente possa vir a ser selecionada. Quando uma operação é selecionada para ser executada, ela é removida da lista de invocações pendentes da operação. A operação é então executada de acordo com os parâmetros atuais da invocação selecionada. Quando a execução da operação termina, também termina a execução do comando SELECT. Aí, então, o kernel verifica o tipo de invocação que foi atendido. Se a invocação foi feita por comando SEND a área alocada para passagem de parâmetros é devolvida. A resposta de uma invocação feita por comando CALL faz com que o processo invocante seja despertado caso a invocação tenha sido de um comando CALL isolado ou caso todas as invocações de um comando CONC já tenham sido atendidas.

O processo que esteve bloqueado por causa de uma invocação síncrona (CALL ou CONC), depois que é escalonada e assume o processador deve recuperar os resultados da execução das operações que ele invocou. Se os resultados se encontram em áreas que foram alocadas dinamicamente, estas áreas devem ser devolvidas. Isto completa a execução de um comando CALL ou CONC, e o processo continua executando a partir do seu próximo comando.

Gerenciamento de Áreas de Armazenamento de Parâmetros de Operações

Os processos de um programa Constructor trocam informações entre si através dos parâme-

tros que são passados quando operações são invocadas e quando operações são executadas com produção de resultados (resposta). Para que estes parâmetros sejam passados entre os processos é necessário que áreas para armazenamento dos valores desses parâmetros sejam usadas. Estas áreas são preenchidas quando invocações de operações que passam parâmetros são realizadas, e são atualizadas depois que operações que produzem resultados são executadas. O fornecimento dessas áreas de armazenamento de parâmetros é feito pelo kernel.

Dois tipos de áreas são supridos: áreas permanentes e áreas dinâmicas. As áreas permanentes são aquelas alocadas para o processo quando da sua inicialização. Cada processo, ao ser inicializado, é associado a uma área fixa para passagem de parâmetros. Esta área será utilizada para os parâmetros das chamadas CALL-simples (não embutidas em comandos CONC). As áreas dinâmicas são alocadas durante a execução dos processos de acordo com a necessidade. A alocação de uma área dinâmica é solicitada quando é executado um comando SEND ou um comando CALL embutido em comando CONC, que necessitem passar parâmetros.

O kernel possuirá uma área para atender às solicitações de alocação. A alocação de áreas é feita durante a inicialização de um processo, alocação de área permanente, ou quando solicitado por um processo, alocação de área dinâmica. Se uma área for alocada por um comando SEND ou por um comando CALL-embutido, ela ficará associada temporariamente ao processo.

Manipulação de Interrupções e Execução de Operações de Entrada/Saída

Os mecanismos específicos para atendimento de interrupções e operações de E/S são dependentes de máquina.

Como a linguagem Constructor é destinada também ao desenvolvimento de software básico, é conveniente que os mecanismos utilizados para sincronização e intercomunicação de processos possam ser usados para o atendimento de interrupções e execução de operações de E/S. O kernel deverá possibilitar que as ocorrências de interrupções possam ser mapeadas em invocações de operações e o compilador deverá possibilitar que variáveis do programa possam ser alocadas em posições absolutas de memória associadas a dispositivos de E/S. Por motivos de segurança estes recursos da linguagem só deverão ser permitidos dentro de módulos especiais.

Para permitir que interrupções possam ser transformadas em chamadas de operações deve existir no kernel uma rotina que transforme ocorrência de interrupções em invocação de operações. O endereço dessa rotina deve ser colocado em todas as entradas associadas a interrupções. Esta rotina deve ter acesso a uma tabela que indique, para cada interrupção, qual é a operação a ser invocada. Quando esta rotina recebe o controle é sinal que ocorreu uma interrupção para qual existe operação associada. Após descobrir que interrupção ocorreu, a rotina simula a ocorrência de uma invocação SEND, sem parâmetros, para a operação correspondente.

Para implementar operações de E/S em um dispositivo, por exemplo, o programador associa

variáveis aos registradores de estado, de comando, e de dados desse dispositivo e associa uma operação às interrupções respectivas. Atribuindo valores às variáveis associadas aos registradores de comando e de dados consegue-se iniciar a operação do dispositivo.

Quando acabar a operação de E/S comandada (isto é, quando acontecer a interrupção do dispositivo), irá ocorrer a invocação da operação associada, a qual poderá obter o resultado da operação de E/S executada consultando os valores das variáveis correspondentes aos registradores de estado e de dados do dispositivo.

Convém observar que a existência destes recursos possibilita ao programador da linguagem de alto nível Constructor o domínio dos sistemas de interrupções e de E/S.

ESTRUTURAS DE DADOS DO KERNEL

O kernel para fornecer um ambiente de execução Constructor deve manter estruturas de dados que permitam a implantação de processos e de operações. Estas estruturas auxiliarão o kernel no fornecimento das unidades de processamento virtuais para todos os processos do sistema componente. Além disso, através delas, será fornecido o suporte para que processos interajam e para que interrupções sejam atendidas.

As principais estruturas de dados são: os descritores de processos, as áreas de parâmetros, as listas de invocação de operações, as filas de processos, o indicador de urgência e a tabela de mapeamento de interrupção em operação.

PRIMITIVAS DO KERNEL

As funções básicas primitivas do kernel são relacionadas a seguir. Elas são: primitiva de inicialização, primitivas para manipulação de áreas para parâmetros de operação, primitivas correspondentes aos comandos de interações e primitivas para controle de processos.

A primitiva de inicialização de processos *inieproc* é invocada durate a fase de inicialização do sistema. Para que um processo seja completamente criado é necessário que sejam alocadas para ele um registro descritor de processo, as cabeças de filas de operações e a área permanente para passagem de parâmetros. O registro descritor é então inicializado e passa a representar o processo perante o kernel.

As primitivas para manipulação de áreas para parâmetros de operações *allocate* e *deallocate* são usadas para solicitar e devolver, respectivamente, áreas para armazenamento de parâmetros de operações. Durante a inicialização dos processos a primitiva *allocate* é invocada para fornecer áreas permanentes para armazenamento de parâmetros de operações. Quando os processos estiverem em execução, a primitiva *allocate* será invocada quando um comando SEND ou CALL-embutido for executado ou quando uma invocação remota para uma operação local for recebida pelo kernel, para que seja fornecida uma área dinâmica para armazenamento de parâmetro de operações. A primitiva *deallocate* é invocada para que uma área dinâmica previamente alocada seja devolvida ao kernel.

As primitivas correspondentes aos comandos de interação são: a primitiva *invoke* , a primitiva *endconc*, a primitiva *delay*, a primitiva *invrem* e a primitiva *reply*. A primitiva *invoke* é usada pelos comandos CALL e SEND para solicitar que o kernel encaminhe a invocação de uma operação ao processo que a implementa. A primitiva *endconc* é utilizada para sinalizar a finalização de um comando CONC. A primitiva *delay* é invocada para retardar a execução de um comando SELECT por não ser possível fazer uma seleção de operação a executar e o comando não possuir cláusula ELSE. Quando uma operação é selecionada para execução, a invocação premiada com a seleção deve ser removida da lista de invocações pendentes e isto é feito através da invocação da primitiva *invrem*. A primitiva *reply* é invocada depois que a execução de uma operação é concluída. Ela é usada para avisar que a invocação da operação foi atendida.

As primitivas para controle de processos são: a primitiva *suspend*, a primitiva *activate*, a primitiva *restart*, a primitiva *kill* e a primitiva *quit*. Estas primitivas permitem que processos exerçam controle sobre outros processos (processos controlados).

SIMULAÇÃO DO KERNEL DISTRIBUÍDO EM AMBINETE CENTRALIZADO

Visando um estreitamento na relação com esses novos conceitos de programação, buscou-se implementar um ambiente onde experiências possam ser realizadas. O ambiente escolhido foi o do sistema centralizado DEC-10 da UFPE. Neste ambiente estão sendo simulados os processos e o kernel distribuído. Os processos e o kernel estão programados em Pascal e se utilizam das Facilidades de Comunicação Inter-Processos (IPCF-Inter-Process Communication Facility) do DEC-10 para simular as facilidades de comunicação do sistema. O IPCF permite que processos se comuniquem uns com os outros através do envio e da recepção de pacotes de informações.

Cópias do kernel componente são carregadas no sistema e interconectadas através do IPCF, compondo, assim, o kernel distribuído. Sobre este kernel são clocados os processos que simulam as interações processo-kernel que são realizadas durante a execução de um programa distribuído.

Com este experimento podem ser testados os novos conceitos introduzidos na linguagem Constructor.

CONCLUSÕES

O kernel projetado pode ser implementado com relativa facilidade, tornando disponível ferramentas poderosas para a construção de sistemas distribuídos. Estas ferramentas podem ser utilizadas a partir de versões mais simples do sistema Constructor, que possam ser mais facilmente implementáveis (o sistema Constructor na sua versão mais completa requererá, certamente, um enorme esforço para o seu desenvolvimento). Este kernel pode ser aproveitado a nível de programação assembler, independentemente de qualquer suporte de linguagem de alto nível.

A disponibilidade de uma implementação do sistema, por mais simplificada que fosse a versão implementada, possibilitaria o desenvolvimento de vários tipos de experiências orientadas a aplicações distribuídas (mesmo que o equipamento não fosse fisicamente distribuído, as aplicações seriam logicamente distribuídas).

O trabalho de implementação está se desenvolvendo em um equipamento tradicional (não distribuído). Ainda assim o trabalho é válido pois permite o desenvolvimento de experiências com esse novo estilo de programação baseado no conceito de objetos e operações remotas.

REFERÊNCIAS

(AND 82) ANDREWS, Gregory R. The Distributed Programming Language SR-Mechanisms, Design and Implementation. *Software - Practice and Experience*, 12(8):719-53, Aug. 1982.

(BUZ 84) BUZIN, Paulo F.W.K. *Uma Abordagem para a Concepção e Implementação de Sistemas Distribuídos*. Porto Alegre, PGCC-UFRGS, 1984.

(JON 78) JONES, Anita K. The Object Model: A Conceptual Tool for Structuring Software. In: ADVANCED COURSE ON OPERATING SYSTEMS, MÜnche, Mar. 29-Apr.6, 1978. Berlin, Springer-Verlag, 1978. Cap. 2A. p.8-16. (Lecture Notes in Computer Science, 60)

(POP 78) POPEK, Gerald J. & KLINE, Charles S. Issues in Kernel Design. In: ADVANCED COURSE ON OPERATING SYSTEMS, MÜnche, Mar. 29-Apr.6, 1978. Berlin, Springer-Verlag, 1978. Cap. 3B. p.210-27. (Lecture Notes in Computer Science, 60)

(QUE 84) QUEIROZ, José A.M. de. *Um Modelo de Sistema Distribuído e Proposta de Kernel para a REDURGS*. Porto Alegre, PGCC-UFRGS, 1984.

M. DE GIUSTI*, O. BRIA**, A. DE GIUSTI***

RESUMEN

En este trabajo se presenta la modelización y simulación de una red de procesamiento distribuído en tiempo real, utilizando el lenguaje GPSS.
El tipo de red estudiado involucra N adquisidores remotos, un procesador central, M procesadores locales y M x K ejecutores remotos, conectados a través de un único canal de comunicación, constituyendo un sistema característico en aplicaciones de tiempo real.
La restricción de operar con una arquitectura de "bus" único obligó a estudiar cuidadosamente la simulación del enlace de comunicación a fin de evaluar el protocolo, la velocidad de trasmisión y la longitud de los mensajes (paquetes).
En las conclusiones del trabajo se reflejan los resultados obtenidos utilizando simulación con GPSS para evaluar la perfomance del sistema.

ABSTRACT

This paper presents a simulation model for a class of real time distributed processing network and some results obtained by using the GPSS language.
This type of network studied, which involves N remote data adquisition modules, one central processor, M local processors and M x K remote actuators, connected via an unique communication channel; so representing a characteristic type of system in a real time environment.
The use of a global (multiple access) "bus" determines the most important restriction for system perfomance, so the communication network is carefully simulated to evaluate the protocol to be used, the transmission rate and the packet length.

* Ingeniera Electrónica - CONICET - CeTAD - Facultad de Ingeniería - UNLP - Argentina
** Ingeniero Electrónico - CIC - CeTAD - Facultad de Ingeniería, UNLP - Argentina.
*** Ingeniero Electrónico - CONICET - CeTAD y LAC - Facultad de Ingeniería y Facultad de Ciencias Exactas, UNLP - Argentina.

INTRODUCCION

La importancia de las aplicaciones de tiempo real que involucran sistemas de procesamiento distribuído ha crecido enormemente en los últimos años, constituyendo el estudio y evaluación de performance de estos sistemas (Ref. 1,2,3,4,5) uno de los temas de mayor interés en la literatura en todo el mundo.

La medición de la performance de un sistema de procesamiento distribuído (tiempos de respuesta, confiabilidad, accesibilidad a los recursos, etc.) obliga a concentrar esfuerzos en aquéllos recursos críticos que diferentes procesadores intentan acceder simultáneamente. (Ref. 6,7,8).

Precisamente el tipo de sistemas distribuídos que analizaremos en este trabajo se caracterizan por una arquitectura de "bus" global (Ref. 9,10) en la que el recurso crítico es -- justamente el canal de comunicación.

Las técnicas de evaluación de performance siguen 2 tendencias perfectamente definidas:

** La utilización de modelos analíticos basados en la teoría de colas (Ref. 11,12,13) en los que las mayores dificultades se encuentran en el ajuste del modelo teórico al sistema real y que generalmente no se aconsejan en sistemas complejos cuya arquitectura y carga varía dinámicamente, por la dificil adaptación del modelo a los mismos.

** Los métodos de simulación por computadora (Ref. 14,15,16,17) que históricamente se han empleado con buenos resultados, especialmente utilizando lenguajes orientados. Este - tipo de enfoque no devuelve una solución analítica del sistema, pero permite "medir" - con un ajuste que depende de la bondad del modelo,la performance del sistema punto a - punto.

Nuestro "problema tipo", tal como se detalla en el punto siguiente, es una configuración altamente variable y con un recurso crítico cuya carga depende de los mensajes ("paquetes") en curso en todo el sistema. Por otra parte existían una serie de parámetros tales como la velocidad de trasmisión, la longitud máxima permitida de cada paquete y el protocolo a adoptar que debíamos definir en función de los resultados de la evaluación de performance. Por ello creímos que lo más conveniente era utilizar simulación.

Por otra parte la Universidad disponía del lenguaje GPSS (General Purpose Simulation System) cuyas prestaciones (Ref. 18) creemos muy adecuadas para este tipo de problemas, por lo que se lo utilizó en el trabajo.

DESCRIPCION DEL TRABAJO

Tal como se ve en la Figura 1, se debía analizar la performance de un sistema distribuído en el que aparecen 4 jerarquías funcionales:

** AIi con $1 \leq i \leq N$ representan adquisidores inteligentes de datos remotos a trasmitir en el sistema. N es variable.

** PLi con $1 \leq i \leq M$ representan procesadores locales que suministran información de -- control a los ejecutores Ei, relacionada con los datos adquiridos por los AI.

** PC representa el procesador central del sistema, que en principio centraliza la fun-- ción de recolección de datos y su distribución, pero que puede desaparecer de la configuración, obligando a una adaptación de la red.

** Ei con 1 ≤ i ≤ K representa el conjunto de ejecutores asociado a un procesador lo-- cal PLj determinado. El número K es variable, aunque será el mismo para todos los PLi.

La información fluye en el sistema por diferentes nodos:

** Entre los AIi y el PC. (mensajes-comandos).

** Entre el PC y los PLi. (información pre-procesada - información procesada).

** Entre los PLi y los Ei. (órdenes - información de acciones).

** Entre los AIi y los PLi. (mensajes-comandos).

El enlace de comunicaciones del sistema se hace sobre un sólo canal, lo que convierte a éste en el recurso crítico más importante a estudiar en la simulación, en particular:

** Evaluación del protocolo a utilizar.

** Estudiar las "colisiones" de mensajes y la técnica para minimizar su efecto.

** Analizar la detección y corrección de errores en la trasmisión.

** Determinar los efectos de diferentes velocidades de trasmisión y diferentes longitudes de mensaje sobre la performance global del sistema.

Si observamos globalmente el esquema de la Figura 1 y las funciones mencionadas anteriormente, se trata de un sistema realimentado (Fig. 2) en el que los AI recogen información de objetos externos al sistema y la trasmiten al PC como novedades; en el PC se lleva un control del estado de cada objeto externo y en función de él (en particular de su ubicación)le asigna su atención a un PLj determinado que procesa la información y activa alguno o todos sus Ei para actuar sobre el objeto.

Esto nos marca el flujo inicial de datos: AI-PC-PL-Ei.

La acción de los Ei correspondientes es informada al PLj asignado y simultáneamente se espera un nuevo mensaje de corrección del AI correspondiente. Lógicamente la función de procesamiento de los PL incluye una estrategia de control a fin de llevar a los objetos ex-- ternos a un "estado" final deseado.

Ejemplos de sistemas distribuídos tal como el descripto lo constituyen los emplazamientos de artillería terrestre, los controladores globales de tráfico de una ciudad, etc. En --- ellos el problema crítico es la evaluación del canal de comunicación elegido y la técnica de protocolización de mensajes que se adopte.

En nuestro caso hemos desarrollado un modelo de evaluación de la red, con GPSS.

DEFINICION DE LA TOPOLOGIA DE LA RED, TRAFICO

En el sistema descripto se identifica claramente un subsistema de comunicaciones. Es decir una red de comunicaciones entre los distintos nodos que lo conforman (ver Fig. 3).

La topología de esta red se eligió conforme a:

1) Ciertas limitaciones técnicas impuestas al sistema:

a) El canal de comunicaciones utilizado es de radiofrecuencia.

b) Se requiere simplicidad operativa para los tranceptores a ser usados en los distintos nodos.

2) El alto grado de conectividad requerido por el sistema según se vió en la definición del mismo. Dando como resultado la elección de un sólo canal de comunicaciones de radiofrecuencia de acceso múltiple. O sea una topología de "bus" compartido o global -- donde el bus es una frecuencia única de trabajo (ver Fig. 4).

Para el sistema propuesto cada Objeto genera una Tarea que consta de un conjunto de Procesos que van realizando en los distintos nodos del sistema. Estos procesos pueden desarrollarse en forma secuencial (uno tras otro) o en forma concurrente (dos o más de ellos en forma simultánea).

El próximo paso sería relevar las características y modalidades del tráfico delineado anteriormente. Estas dependerán indudablemente de la aplicación particular, pero deberán hacerse relevamientos de, por ejemplo:

1) Tipo de mensajes.
   a) Referida a los objetos
   b) De control (comandos)
2) Características de los mensajes a transmitir por cada nodo, por ejemplo: destinos, -- formatos, longitud, etc.
3) Distribución de los tiempo de ocurrencia de las variables de entrada al sistema.
4) Es útil hacer una apreciación a priori y global de las posibles distribuciones de los tiempos de intergeneración de todos los mensajes internodos.

MODELO DEL SISTEMA

El sistema, tal como se lo ha descripto, puede representarse por un conjunto de elementos "estáticos" en la simulación que son los AIi, PLi, PC y Ei y un elemento altamente variable que es el canal de comunicación.

Por esto dividimos el modelo en 2 módulos: aquél que trata los nodos fijos y el tratamiento del canal.

Modelización de los componentes fijos

Los elementos fijos del sistema global, este es: adquisidores, procesadores y ejecutores, se simulan de acuerdo a los siguientes lineamientos generales:

Los objetos se consideran como los elementos a generar en el sistema, es decir las TRANSACCIONES cuyas características: posición, tamaño, prioridad de atención, etc. son almacenadas en parámetros GPSS. Tales transacciones esperarán en una COLA para ser atendidas por la computadora, que, en este aspecto, es una FACILIDAD del lenguaje. Las estadísticas de colas, provistas automáticamente por el lenguaje, brindarán información sobre los retardos de interés.

Los ejecutores, que pueden verse como FACILIDADES alterarán el estado de cada objeto (pa-

ra el lenguaje, los parámetros). El sistema se realimentará así hasta llegar al estado final. Cada transacción al llegar a tal punto será destruída por un bloque específico del lenguaje.

El aspecto más importante que puede variarse en la simulación es la distribución de las apariciones de los objetos y la frecuencia de las mismas. Asimismo, cuando el modelo esté completo, puede estudiarse el número de realimentaciones necesario (para llevar cada objeto a la característica deseada), con un determinado algoritmo de atención y determinada velocidad de deriva de objetos, a fin de establecer los límites del sistema.

Modelización global de la red de comunicaciones

Debido al canal definido anteriormente los distintos nodos del sistema pueden relacionarse topológicamente como se ve en la Figura 5. En cada nodo se identifican claramente tres elementos constitutivos:

1) Transmisor de datos del nodo al canal.
2) Receptor de datos disponibles sobre el canal.
3) Dispositivo procesador de datos del nodo particular.

Los elementos 1 y 2 conforman lo que puede denominarse Unidad de Interfase con el Canal (UIC). Cada dispositivo procesador (PC) queda así conectado a una UIC a través de una vía de comunicación serie o paralelo (dependiendo del dispositivo). La UIC almacena los datos recibidos desde los DP, conforma con ellos paquetes, y los envía al canal (a través de un MODEM) como una cadena de datos, obedeciendo a las reglas fijadas por algún protocolo. Cada UIC está atenta a todas las cadenas de datos sobre el canal, con el objeto de detectar los paquetes destinados al nodo correspondiente. En este punto debe mencionarse que para el sistema propuesto todas las UICs están normalmente en escucha y dejan de estarlo durante la transmisión.

Definición de las características generales del subsistema de comunicaciones

1) La primera característica a ser definida es la técnica de acceso al canal. Se eligió -- la técnica de acceso aleatorio atendiendo a las siguientes consideraciones:
a) No se requiere sincronización entre la UIC de los distintos nodos pues el control de acceso es completamente distribuído.
b) Se supone que las tareas se generan aleatoriamente.
c) Los tiempos de procesamiento pueden en general ser aleatorios.
d) Las características globales del tráfico.
2) La segunda característica a ser definida es la técnica de envío. Pero ésta queda fija-- da naturalmente como broadcasting.
3) El protocolo debe incluir alguna técnica de direccionamiento de los nodos de destino -- y de identificación de los nodos de origen, de cada paquete.

Así los mensajes son enviados desde un nodo a otro de la red de la siguiente manera:

a) Conformando paquetes

b) Agregando información de dirección (y origen) a los paquetes
c) Accediendo al canal a través de una técnica aleatoria
d) Difundiendo los paquetes por el canal
El receptor atento a los paquetes sobre el canal realiza la conmutación tomando los paquetes que corresponden a su dirección.

Modelo de simulación del canal de acceso aleatorio

El primer paso fue definir un modelo conceptualmente simple que pudiera ser validado por comparación con resultados teóricos ya conocidos.
Luego, siguiendo una técnica top-down se fueron desarrollando modelos de complejidad creciente utilizando como semilla el modelo simple (Ref. 19).
Los programas de simulación escritos en GPSS para simular los modelos conceptuales se desarrollaron concurrentemente con éstos.
A) Con la finalidad de validar los modelos a ser definidos se eligió un modelo de tráfico simple de características completamente aleatorias (tiempos de interarrivo con distribución de Poisson).
B) Independientemente del protocolo particular que se utilice, en todo canal de acceso aleatorio, puede producirse el fenómeno de colisión de paquetes. Esto es, cuando dos o más paquetes ocupan al mismo tiempo el canal es muy probable que alguno o algunos de los bits que conforman la información del paquete se modifiquen; esto ocasionará (en el mejor de los casos) la detección de un error cuando se chequee la información del paquete en el receptor.
El primer paso fue, entonces, modelizar un bloque denominado "canal" cuya misión principal es detectar la colisión de paquetes en el medio físico y asignarle la característica de colisionado o no, según se cumpla aquella condición. Estas características del bloque "canal" hacen de él la unidad básica en la simulación de cualquier protocolo de acceso aleatorio (ver Fig. 6).
Para la modelización del bloque canal se hicieron las siguientes suposiciones:
1) Se detectan todas las colisiones, produzcan o no cambios en la información de los paquetes.
2) No se considera el efecto del ruido. Indudablemente este puede producir algún cambio de información sobre algún paquete que no haya colisionado.
3) Los paquetes son todos de igual longitud.
4) Se considera que el canal no introduce retardo.
El bloque descripto modeliza fundamentalmente las características generales de un canal de acceso múltiple del tipo ALOHA (Ref. 20).
C) Se escribió un programa de simulación para el bloque "canal". Este queda básicamente representado en GPSS por un STORAGE; la capacidad de éste debe ser idealmente indefinida; pero para una configuración dada basta con una capacidad limitada.

Uno de los parámetros de las TRANSACCIONES (representando a los paquetes) queda reservado para la asignación de la característica de colisionado (ABT) o no (O.K.).
Los datos simunistrados por los resultados de los programas fueron cotejados con los resultados teóricos (Ref. 21) para validar el modelo simple.
D) Una vez que una colisión ha sido detectada sobre un canal de acceso aleatorio, los nodos participantes deben reesquematizar sus paquetes de forma tal que sean transmitidos un tiempo después.
El próximo paso fue entonces definir, simular y validar un modelo en el cual los mensajes ABT fueran reesquematizados. Para ello fue necesario establecer algoritmos de reesquematización inicial lo suficientemente simple como para poder validar el modelo (Ref. 22).
El algoritmo más simple es aquel en el que se elige un retardo aleatorio. La función de distribución fue de Poisson.
E) En todo protocolo de comunicaciones debe implementarse alguna manera de informar al nodo transmisor si su mensaje fue recibido sin error. En la mayoría de los protocolos una palabra de chequeo de redundancia cíclica es adicionada a cada mensaje (paquete) por el transmisor. Este código es usado por el receptor para asegurarse que el mensaje está libre de errores (debidos por ejemplo a colisiones). Normalmente si el mensaje fue recibido correctamente, el receptor envía un paquete de Reconocimiento (ACK) al transmisor, para que éste quede debidamente notificado. Naturalmente no se espera reconocimiento de los paquetes de ACK.
En el sistema ALOHA se independiza el tráfico de paquetes de mensajes, de aquel destinado al reconocimiento (ACK) de los paquetes recibidos, utilizando dos canales independientes. En nuestro sistema ambos tráficos deben circular por la misma vía de comunicaciones.
Se desarrolló entonces un nuevo modelo que incluyera el tráfico de ACK de los paquetes O.K.
Como era de esperar el throughput del canal disminuyó notablemente.
F) Se reconoció la necesidad de:
1) Adoptar una técnica de acceso al canal del tipo CSMA (Carrier Sense Multiple Access) (Ref. 23).
2) Definir y probar distintos algoritmos de reesquematización.
3) Ver si es posible implementar alguna técnica de priorización del tráfico de ACK.
4) Definir un nuevo modelo de tráfico de prueba de población finita.
5) Fijar algún criterio de priorización del tráfico entrante, del tráfico reesquematizado y el de ACK en los nodos.
Sobre estos y otros tópicos se está trabajando actualmente.

Para la evaluación de la perfomance de un sistema de procesamiento distribuído es necesario estudiar exhaustivamente las características del subsistema de comunicaciones que existe entre los nodos físicos de la red.

La simulación del canal de acceso aleatorio, del tipo de sistema presentado, resulta ser un medio eficaz para definir los parámetros de interés y evaluar su incidencia sobre la performance global.

Esta simulación se ha planteado realizando una estrategia de niveles de complejidad creciente.

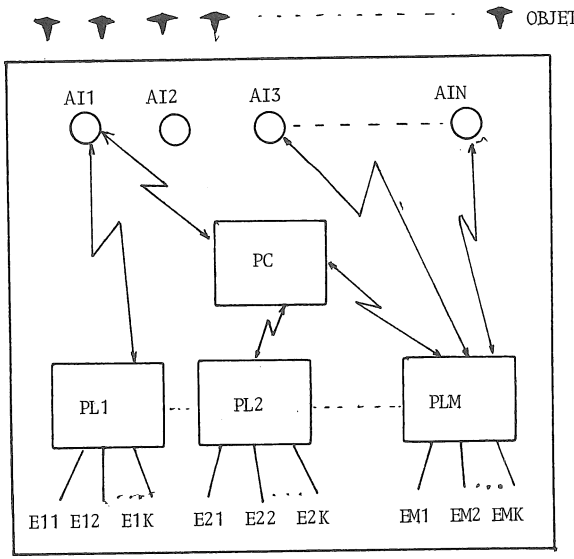


FIGURA 1

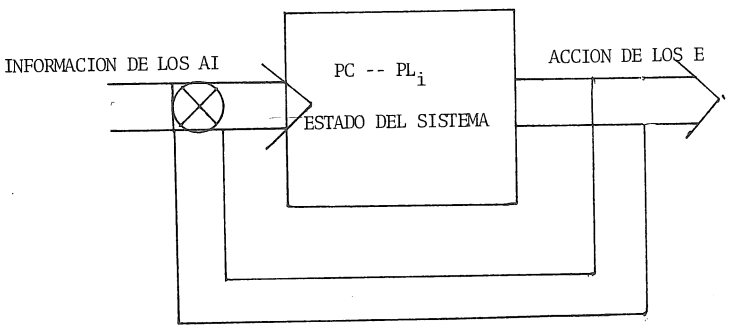


FIGURA 2

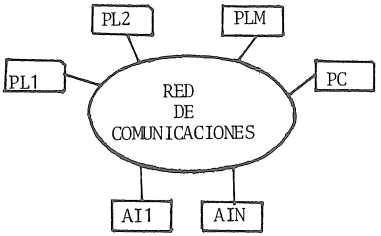


FIGURA 3

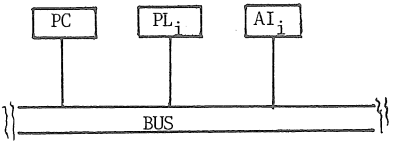


FIGURA 4

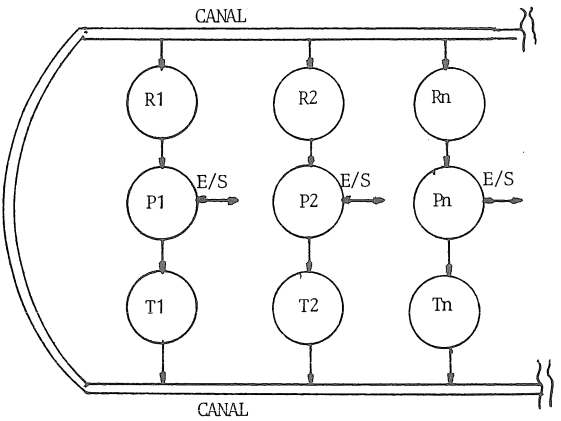
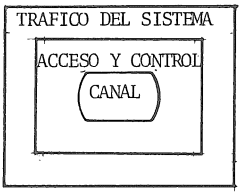
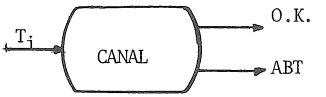


FIGURA 5



(A)



(B)

FIGURA 6

(A) : UBICACION RELATIVA DEL BLOQUE "CANAL" EN EL SISTEMA

(B) : T_i = TRAFICO TOTAL ENTRANTE (PAQUETES NUEVOS Y REESQUEMATIZADOS)

O.K.= PAQUETES QUE NO HAN COLISIONADO

ABT = PAQUETES QUE HAN COLISIONADO

REFERENCIAS

1- "Analysis of Multiple-Microprocessor System Architectures"
A.J. Weissberger (Computer Design, June 1977, págs. 151-163.

2- "Technologies for Local Area Computer Networks"
I.W. Cotton (Proceeding of the LACN Symposium, May 1979, págs. 283-291.

3- "Overview of Synchronization and Recovery Problems in Distributed Databases"
W.H.Kohler (Digest of Papers, COMPOCON 80, Fall 1980, págs. 433-441.

4- "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples"
G.A. Anderson and E.D. Jensen (Computing Surveys, Vol. 7 N° 4, December 1975.

5- "Partitioning and Allocation of Logical Resources in a Distributed Computing Environment".
B.P.Buckles and D.M. Hardin.

6- "Optimal Program and Data Locations in Computer Networks"
H.L. Morgan and K.D. Levin.

7- "Optimal Partitioning of Workload for Distributed Systems"
V.B. Gylys and J.A. Edwards.

8- "Experiences with performance measurement and modeling of a processor array"
H. Fromm. IEEE Trans. on Computers. 1983.

9- "Multiaccess Protocols in Packet Communication Systems"
F.A. Tobagi. IEEE Trans. on Communications, Vol. COM-28, N° 24, April 1980 págs. 468-488.

10- "The Cambridge Digital Communication Ring"
M.V. Wilkes and D.J. Wheeler (Proceedings of the LACN Symposium, May 1979, págs. 47-61.

11- "Analitical queueing models for programs with internal concurrency"
P. Heidelberger IEEE Trans. on Computers. 1983.

12- "Queueing models of computer systems"
A. Allen. IEEE Computer, 1980.

13- "Sistema de Multiprocesamiento de Tiempo Real. Simulación"
A. De Giusti, G. Rossi. 1er. Congreso Latinoamericano de Automática, Brasil, 1984.

14- "Computer system simulation: a tutorial"
M.H. Mc Dougal. Computing surveys ACM, 1970.

15-"A shared resource algorithm for distributed simulation"
P. Reynolds. 9th annual symposium on computer arqutecture.

16- "Simulación de un sistema de multiprocesamiento para adquisición de datos"
G. Rossi. Informe Técnico CeTAD. 1984.

17- "A description of the simscript language"
B. Dimsdale. IBM system journal.

18- GPSS manuals IBM

19- "Integrated Design and Verification of Simulation Programs"
R. Chattergy, U.W. Pooch. IEEE Computer, April 1977.

20- "ALOHA Packet Broadcasting-A restrospect"
R. Rinder, N. Abramson et al. (Proceeding, National Computer Conference, 1975) IEEE Tutorial on Local Computer Network, second edition.

21- "Packet Switching in Radio Chanel: Part I Carrier Multiple-Access Modes and Their Throughput-Delay Characterization"
L. Kleinrock, F.A. Tobagi. IEEE Transactions on Communications, Vol. COM-23 N° 12, December 1975.

22- "Global Bus Computer Communication Techniques"
E.C. Luczak. IEEE, Tutorial on A Pragmatic View of Distributed Processing Systems.

23- "Multiaccess Protocols in Packet Communication System"
F.A. Tobagi. IEEE Transactions on Communication, Vol. COM-28, N° 4, April 1980.

* L.V. TOSCANI
** P.A.S. VELOSO

SUMARIO

A programação dinâmica é uma técnica de desenvolvimento de algoritmos muito útil para uma certa classe de problemas. Este artigo apresenta uma formalização da programação dinâmica através de tipos abstratos de dados e um estudo da complexidade intrínseca dessa técnica.

ABSTRACT

Dynamic programming is a technique for algorithm development which is very useful for a certain class of problems. This paper presents a formalization of dynamic programming by means of abstract data types, as well as a study of the intrinsic complexity of this technique.

* Mestre em Informática (PUC/RJ, 73); desenvolvimento e complexidade de algoritmos; Prof. adjunto da UFRGS; Cx. Postal 1501, 90000, Porto Alegre, RS.

** Ph.D. em Ciências da Comp.(Univ. da Califórnia - Berkeley, 75); teoria e metodologia de programação; Prof. associado na PUC/RJ; Depto. de Informática PUC/RJ; Rua Marques de São Vicente 225, 22.453, Rio de Janeiro, RJ.

1. INTRODUÇÃO

A formalização de métodos computacionais busca seu aproveitamento mais eficaz e abrangente. Além disso, permite uma melhor compreensão e uma maior clareza para os métodos, facilitando o estudo dos seus desempenhos.

Este trabalho descreve um método de desenvolvimento de algoritmos, conhecido por programação dinâmica, apresentando as definições informal e formal, com diagrama sintático, exemplo e um estudo da complexidade de tempo intrínseca do método.

2. APRESENTAÇÃO INFORMAL

A programação dinâmica é um método de desenvolvimento de algoritmos, que consiste em, dado um problema, dividí-lo em subproblemas, solucionar os subproblemas, guardar os subresultados, combinar subproblemas menores e subresultados para obter e resolver problemas maiores, até recompor e resolver o problema original. O diagrama sintático da figura 1 descreve essa situação (P = domínio de problemas, R = domínio de resultados).

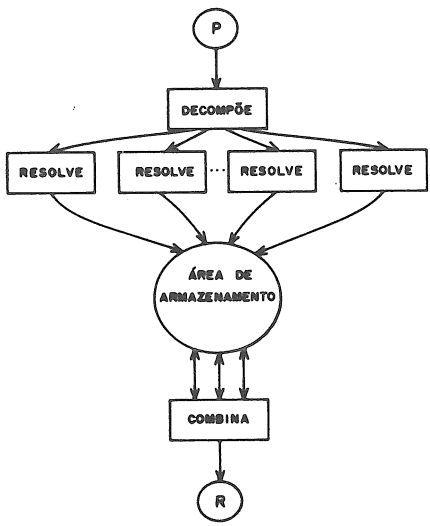


Figura 1

O problema é decomposto uma única vez, assim os subproblemas menores são gerados antes dos maiores. Por esse motivo esse método é chamado ascendente, ao contrário dos metodos recursivos que são descendentes.

3. DOMÍNIO DE APLICAÇÃO

A programação dinâmica é usada para resolver problemas cuja solução pode ser obtida através de uma seqüência ótima de decisões, conforme o diagrama sintático da figura 2.

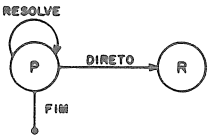


Figura 2

Decisões são tomadas sucessivamente até que uma certa condição fim (solução encontrada) seja atingida, quando é obtida a resposta final.

A seção seguinte apresenta um exemplo de aplicação desta técnica. Outros problemas que podem ser resolvidos por programação dinâmica são: árvore binária de busca ótima, problema do caixeiro viajante, etc |TER 82|.

4. EXEMPLO ILUSTRATIVO

Em |TER 82| é apresentado um exemplo simples que caracteriza bem o método. Esse exemplo será transcrito aqui.

Deseja-se multiplicar n matrizes, i.é, calcular $M=M_1 x\ M_2 x \ldots x M_n$ onde cada matriz M_i tem b_{i-1} linhas e b_i colunas $1 \leqslant i \leqslant n$. O algoritmo trivial requer p x q x r operações para multiplicar uma matriz p x q por outra q x r. A multiplicação de matrizes é associativa, existem portanto várias maneira possíveis de se realizar essa multiplicação com diferentes números de operações correspondentes. O problema consiste, então, em determinar a seqüência de multiplicações que requer o número mínimo de operações.

Um algoritmo que enumere todas as seqüências possíveis, calcule os respectivos números totais de operações requeridas e em seguida escolha a seqüência ótimo, tem complexidade exponencial em n (número de matrizes), o que é inviável na prática, quando n é relativamente grande.

O algoritmo apresentado aqui, num primeiro passo decompõe o problema em n subproblemas de tamanho 1, resolve-os guarda o resultado na diagonal de uma matriz; passo a passo combina as soluções de n-u+1 problemas de tamanho u, para resolver n-u problemas de tamanho u+1 (u=1,2,...,n-1) e guarda os resultados numa diagonal superior da matriz. O processo termina com a solução de 1 problema de tamanho n.

Chame de m_{ij} o custo do produto, isto é o número mínimo de operações necessárias para calcular $M_i \times M_{i+1} \times \ldots \times M_j$. Se $M' = M_i \times \ldots \times M_k$ e $M'' = M_{k+1} \times \ldots \times M_j$ o custo mínimo de $M' \times M''$ é $m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j$, o custo mínimo de $M_i \times M_{i+1} \times \ldots \times M_j$ é $m_{ij} =$

$$\min_{i \leq k < j} (m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j).$$

Entrada: $(b_0, b_1, \ldots, b_n)$

Saída: m_{1n}

```
1. para i ← 1 até n faça m_ii ← 0 fim-para;
2. para u ← 1 até n-1 faça
3.       para i ← 1 até n-u faça
4.            j ← i+u    (* u = j-1 *)
5.            m_ij ← min (m_ik + m_k+1,j + b_i-1 x b_k x b_j)
                      i≤k<j
6.            fim-para
7.       fim-para
8. para-com-saída (m_in)
```

A seqüência ótima das multiplicações pode ser obtida guardando os valores k obtidos na linha 5 do algoritmo.

A complexidade do algoritmo é $O(n^3)$; a programação dinâmica tornou possível resolver o problema com uma ordem de complexidade polinomial.

5. FORMALIZAÇÃO DO MÉTODO

Nesta secção a programação dinâmica é definida formalmente como um tipo de dado abstrato.

5.1 Diagrama Sintático

Domínios: P, domínio das instâncias de problemas, R, domínio de resultados, N, conjuntos dos números naturais, Q, conjunto das seqüências de elementos de P, M, conjunto das seqüências de elementos de R.

$Q = P^+ = \bigcup_{i \in N-\{0\}} P^i$ $\qquad$ $M = R^+ = \bigcup_{i \in N-\{0\}} R^i$

Funções: decompõe (P → Q), decompõe o problema original em subproblemas de tamanho mínimo; inicializa (Q → M), resolve os problemas de tamanho mínimo; combina (Q x M → Q) combina os problemas atuais para criar um novo conjunto de problemas de tamanho uma unidade maior; atualiza (Q x M → M), resolve os problemas atuais, usando as soluções dos problemas do nível anterior; recupera (M → R), recupera a solução do problema; tamanho (P → M), dá o tamanho do problema.

Predicados: resolve (P x R), resolve (p,r) = r é solução de p; subinst (P x P), subinst (p, q) = q é subinstância de p.

A partir dessas funções e predicados foram definidos a função tamanho$^+$ e o predicado resolve$^+$, como segue: tamanho$^+$: Q → N, tamanho$^+$ (p_1, p_2,..., p_i) = max {tamanho (p_1),..., tamanho (p_i)} e resolve$^+$ (PxR)$^+$, resolve$^+$ ((p_1, p_2,...,p_i), (r_1, r_2,...,r_i)) = $\bigwedge_{j=1,\ldots,i}$ resolve (p_j, r_j), onde $(PxR)^+ = \bigcup_{i \in N-\{0\}} P^i \times R^i$, subinst$^+$(p,($q_1$,...,$q_i$)) = $\bigwedge_{j=1,..i}$ subinst(p,q_j)

O relacionamento entre os domínios funções e predicados é mostrado na figura 3.

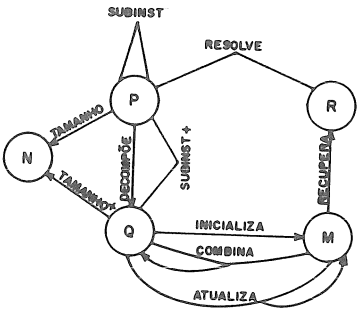


Figura 3

5.2 Axiomatização

Os seis axiomas seguintes definem a semântica do tipo

AX1. $(\forall p)$ (resolve$^+$ (decompõe (p), inicializa (decompõe (p))))

AX2. $(\forall q)$ $(\forall m)$ [resolve$^+$ (q,m) → resolve$^+$ (combina (q,m), atualiza (q,m))]

AX3. $(\forall p)$ subsinst (p, decompõe (p))

AX4. $(\forall p)$ $(\forall q)$ $(\forall m)$ [subinst$^+$(p,q) → subinst$^+$(p, combina (q,m))]

AX5. $(\forall q)$ $(\forall m)$(tamanho$^+$ (combina (q,m)) = tamanho$^+$ (q)+1)

AX6. $(\forall p)$ $(\forall q)(\forall m)$[(subinst$^+$(p,q) $\wedge$ tamanho (p) = tamanho$^+$ (q) $\wedge$
resolve$^+$ (q,m)) → resolve (p, recupera(m))]

Observações:

O axioma 1 diz que a função inicializa calcula os resultados dos subproblemas resultantes da decomposição. No axioma 2 é dito que as funções combina e atualiza mantêm a relação resolve. A função decompõe, decompõe uma instância de um problema em subinstância do problema inicial, é o que diz o axioma 3. Pelo axioma 4 a função combina mantêm a relação subsinstância. O axioma 5 define o tamanho do problema gerado pela função combina, como uma unidade maior que o tamanho do problema que lhe deu origem. E pelo axioma 6 toda subinstância q de mesmo tamanho do problema inicial p é equivalente ao problema p, no sentido de que a solução do problema q implica na solução do problema inicial p.

6. PROGRAMA ABSTRATO

Nesta secção a programação dinâmica é apresentada como um programa abstrato, utilizando as funções auxiliares já definidas e as seguintes variáveis:

p - problema entrada
r - solução do problema
n - tamanho da entrada
q - seqüência de problemas parciais
m - seqüência de soluções parciais
mínimo - tamanho dos problemas resultantes da decomposição
k - tamanho dos problemas atuais

Programa:

1. entrada: p
2. n ← tamanho (p)
3. q ← decompõe (p)
4. mínimo ← tamanho$^+$ (q)
5. m ← inicializa (q)
6. para k = mínimo até n-1 faça

```
 7.      q,m ← (combina, atualiza) (q,m)
 8.      k ← k+1
 9. fim-para
10. r ← recupera (m)
11. saída: r
```

Nas linhas 1 a 5 são inicializadas as variáveis p,n,q, mínimo e m. A variável mínimo recebe o valor do tamanho dos problemas q, resultantes da decomposição da entrada. As linhas de 6 a 9 contêm uma iteração que é executada exatamente n-mínimo vezes. A variável k guarda o tamanho dos problemas que estão sendo resolvidos. Cada iteração resolve a seqüência q de problemas, guarda os resultados em m e cria uma nova seqüência q de problemas de tamanho k + 1. Na linha 10 é recuperado o resultado final.

A verificação da correção do programa pode ser encontrada em |TOS 85|.

7. APLICAÇÃO AO EXEMPLO

Usando esse formalismo o exemplo anterior toma a seguinte forma:
Tipos de dados: P, conjunto de seqüências de matrizes reais multiplicáveis; R = N; $Q=P^+$.

Variáveis: $p = < M_1,\ldots, M_n > \varepsilon P$; n, constante inteira cujo conteúdo é o número de elementos de p; m, matriz n x n.

Funções: tamanho, número de matrizes consideradas no problema; decompõe, decompõe o problema de tamanho n em n problemas de tamanho 1, isto é ao invés de considerar o produto $M_1 \times M_2 \times \ldots \times M_n$ são consideradas as matrizes $M_1, M_2, \ldots M_n$; inicializa, inicializa a diagonal da matriz m com zeros; combina considera todos n-u produtos parciais de tamanho u + 1 i. é, $M_i \times M_{i+1} \times \ldots \times M_{i+u}$ $i=1,\ldots, n-u$; atualiza, preenche a próxima diagonal superior da matriz da seguinte maneira $m_{ij} \leftarrow \min_{i \leqslant k < j} (m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j)$ para $i=1,2,\ldots, n-u$ e $j = i+u$; recupera, recupera o valor contido em m_{1n}.

Predicados: resolve (p,r), r é o número de operações necessários para multiplicar as matrizes de p; subsinst (p,q), todo elemento de q é uma subinstância de p "sem buracos", isto é se $p = < M_1, M_2, \ldots, M_n >$ um elemento de q é uma seqüência do tipo $< M_i, M_{i+1}, \ldots, M_{i+k} >$

Comparando os programas da secção 4 e secção 6 verifica-se a seguinte correspondência.

programa da secção 4

Entrada $(b_0, b_1, \ldots, b_n)$

1. para i ← 1 até n faça $m_{ij} \leftarrow 0$ fim-para
2. para u ← 1 até n-1 faça
3. para i ← 1 até n-u faça
4. j ← i+u
5. $m_{ij} \leftarrow \min_{i \leq k < j} (m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j)$
6. fim-para
7. fim-para
8. para com saída (m_{1n})

programa da secção 6

1. entrada: p
2. n ← tamanho (p)
3. q ← decompõe (p)
4. mínimo ← tamanho$^+$ (q)
5. m ← inicializa (q)
6. para k= mínimo até n-1 faça
7. q, m ← (combina, atualiza) (q,m)
8. k ← k+1
9. fim-para
10. r ← recupera (m)
11. saída: r

A verificação de que o exemplo satisfaz os 6 axiomas pode ser encontrado em |TOS 85|

8. COMPLEXIDADE DE TEMPO INTRÍNSECA DO MÉTODO

A complexidade das linhas 2 a 5 é a soma das complexidades de cada linha, que depende do tamanho n do problema cnsiderado e varia com as funções (tamanho, decompõe, tamanho$^+$, inicializa) executadas. As linha 6 a 9 definem uma iteração de n-mínimo passos. O tempo de uma execução dessa iteração depende do tamanho k dos problemas considerados e da-complexidade da linha 7. A complexidade da iteração é a soma das complexidade da cada execução. A complexidade da linha 7 é o número de problemas da seqüência q vezes a complexidade de solução de um problema de tamanho k.

Usando a seguinte notação: c(g) para complexidade da função g e Comp para complexidade total do algoritmo conclui-se:

$$Comp = c\ (tamanho) + c\ (decompõe) + c\ (tamanho^{+}) + c\ (inicializa) + \sum_{k=mínimo}^{n-1} c\ (combina, atualiza) + c\ (recupera)$$

Fazendo algumas suposições com respeito à complexidade das funções primitivas é possível obter-se algumas regras de comportamento do método.

Suponha que as funções tamanho, decompõe, tamanho^{+} e recupera tem complexidade constante e a função inicializa tem complexidade de O (n) (Esta suposição é bem razoável!). Pode-se ser um pouco mais geral e supor que a soma das complexidades das cinco funções em questão é de O (n). Partindo desta suposição a complexidade do algoritmo fica:

$$Comp = c'n + \sum_{k=mínimo}^{n-1} c(combina, atualiza)$$

Além disso, suponha que mínimo = 1 e que c(combina, atualiza) é polinomial em k. Então tem-se $Comp = c'n + \sum_{k=1}^{n-1} k^{i}$, mas esse somatório é $O(n^{i+1})$, logo a complexidade do algoritmo é polinomial e de ordem i+1 |TOS 85| (mínimo = 1 é irrelevante e não altera a complexidade total).

Por outro lado, se c(combina, atualiza) é não polinomial $\sum_{k=mínimo}^{n-1} c(combina, atualiza)$ é não polinomial e a complexidade total do algoritmo é também não polinomial.

Para o exemplo apresentado tem-se: c(tamanho) é constante, c (tamanho^{+}) é constante, c (inicializa) é O(n), c (recupera) é constante, c (combina, atualiza)) é O((n-k) x k), mínimo=1. Logo,

$$Comp = c_1' n + \sum_{k=1}^{n-1} ((n-k) \times k) = O(n^3)$$

9. CONCLUSÃO

A programação dinâmica é especialmente útil no desenvolvimento de algoritmos, quando a solução do problema pode ser alcançada após uma seqüência de decisões. A seqüência ótima de decisões é obtida evitando as seqüências que sabidamente não resultam na seqüência ótima.

A abstração focaliza alguns aspectos do problema, considerados mais importantes e obscurece os detalhes menos importantes para o enfoque desejado. A abstração do método apresentado nesse trabalho evidencia o processo iterativo ascendente e o aspecto de armazenamento de resultados parciais, mas obscurece o processo de rejeição de seqüências de deci-

sões que sabidamente não resultariam na seqüência ótima. Esse enfoque foi muito útil no estudo da complexidade intrínseca da programação dinâmica. Outra abstração do método que evidencia outros aspectos pode ser encontrada em |WAG 84|.

A programação dinâmica é não só uma técnica a mais de desenvolvimento de algoritmos, mas uma técnica valiosa no desenvolvimento de algoritmos eficientes. É especialmente útil no projeto de algoritmos para problemas NP-difíceis, como o problema do caixeiro viajante, por exemplo, cujo algoritmo trivial é $O(n!)$ e o algoritmo obtido através dessa técnica é $O(n^2\ 2^n)$ |TER 82|e|TOS 85|. Outros exemplos de algoritmos para problemas NP-difíceis obtidos pela programação dinâmica podem ser encontrados em |HOR 78|.

Outros métodos de desenvolvimento de algoritmos foram estudados e formalizados em |VEL 80| e |WAG 84|.

BIBLIOGRAFIA

|AHO 74| - AHO, A.V., J.E. HOPCROFT & J.D. ULLMAN. "The Design and Analysis of Computer Algoritms". Reading, Mass., Addison-Wesley, 1974.

|GRE 81| - GREENE, D.H. & D.E. KNUTH. "Mathematics for the Analysis of Algorithms". Boston, Birkhauser, 1981.

|HOR 78| - HOROWITZ, E. & S. SAHNI. "Fundaments of Computer Algorithms". Comp. Sci. Press. 1978.

|TER 82| - TERADA, R. "Desenvolvimento de Algoritmos e Complexidade de Computação". Terceira Escola de Computação. Depto. de Informática, PUC/RJ, 1982.

|TOS 85| - TOSCANI, L.V. & VELOSO, P.A.S. Programação Dinâmica via tipos abstrato de dados. Porto Alegre, PGCC da UFRGS. (a publicar).

|VEL 80| - VELOSO, P.A.S. "Divide - and - Couquer via Data Types". Proc. III Latin-American Conf. on Informatics, Caracas, Venezuela, 1980.

|VEL 83| - VELOSO, P.A.S. "Problems and Solutions: Towards a General Formulation" - 3rd International Conference in Computer Science. Santiago, Chile, 1983.

|WAG 84| - WAGA, C.F.E. "Métodos para resolver problemas". Depto. Informática da PUC/RJ, Dissertação de mestrado. Rio, 1984.

A.R. CAVALLI*

RESUMEN

En este trabajo se define un método de decision para la logica temporal lineal. Este método se define como una extension del principio de resolucion a los operadores temporales y posee la forma de un conjunto de reglas recursivas que estan basadas en inferencias tautologicas de la logica temporal.

Se realiza una comparacion con el método de los "tableaux semanticos" dando el grafo correspondiente al "tableau" y el arbol de resolucion que se obtiene aplicando el método de resolucion definido por nosotros, mostrando el caracter semantico del primero y sintactico del segundo. La aplicacion de los tableaux semanticos a la verificacion de propiedades de vivacidad ("liveness properties") de los programas, exige establecer ciertas restriccionnes sobre los modelos, mientras que el método de resolucion que opera por transformaciones de las formulas no plantea esta limitacion.

* Docteur d'Etat es Sciences (Université Paris VII, 1984), logicas y modelos para la especificacion y verificación de sistemas concurrentes, Escuela de Computación - Universidad Central de Venezuela (U.C.V.), Caracas, y actualmente LITP (Laboratoire d'Informatique Théorique et Programmation), Université de Paris VII, 2 Place Jussieu, 75251 PARIS Cedex 05.

La logica temporal puede ser considerada como un instrumento adecuado para la especificación de programas paralelos.

En efecto, en los programas paralelos, no se puede obtener el resultado final del calculo a partir del analisis de las valores de entrada-salida. La ejecución en paralelo puede producir interferencias que modifican el resultado final del calculo, o sea, que el analisis de los programas concurrentes debe tener en cuenta las secuencias de ejecución.

La logica temporal que describe situaciones que cambian en el tiempo, permite formalizar toda la ejecución de un programa y no solamente la relación de entrada-salida.

La interrelación de los programas paralelos plantea nuevos problemas a resolver : la ausencia de bloqueo de los procesos, ya sea bajo la forma de ausencia de bloqueo del sistema global o de cada uno de los procesos ; la sincronización de los procesos, la progresión del sistema, etc...

Surge entonces la necesidad de crear lenguajes que permitan la expresion de este tipo de problemas.

Es para dar respuesta a los problemas planteados por la especificación y verificación de programas paralelos que la logica temporal ha sido estudiada bajo tres aspectos :

- como instrumento de expresion y de descripción de propriedades de programas. Se trata de responder a la pregunta : el lenguaje de la lógica temporal permite expresar de manera exhaustiva todo lo que es verdadero en un modelo lineal arbitrario ?

Dar una respuesta a dicha pregunta es de primera importancia ya que podemos representar la ejecución de un programa como una secuencia de estados.

- como instrumento deductivo

Se trata de responder à la pregunta : Tal propriedad invariante o eventual, expresada por la formula ϕ de la logica temporal, es verdadera para un programa dado ?

La respuesta lleva a desarrollar sistemas deductivos para la lógica temporal, como los desarrollados por Owicki-Lamport, Manna-Pnueli, etc... Estos sistemas deductivos son, en general, completos y consistentes.

-como instrumento de décision

En este caso la pregunta es : la formula ϕ que expresa la especificacion de un programa, posee un modelo ?

Los métodos que han sido desarrollados para dar una respuesta a esta pregunta se basan en métodos de decisión que han sido desarrollados para la lógica clasica . Estos métodos se definen ya sea aplicando un punto de vista "sémantico", es decir, tratando de construir un modelo de la formula ϕ. Este enfoque ha dado lugar a los "semantic tableaux" como método de decision. O aplicando un punto de vista "sintáctico", es decir, aplicando operaciones sobre la fórmula que preservan su satisfabilidad. Si obtenemos la expresión vacia, entonces la formula es insatisfacible, dado que la expresión vacia no posee modelo.

Este enfoque, que se basa en el principio de resolución para la logica clásica, nos ha permitido definir el método de decisión que vamos a exponer en este trabajo y que vamos a comparar con el método de los "semantic tableaux".

El método se define como una extensión de la resolución clásica a los operadores temporales y posee la forma de un conjunto de reglas recursivas. Este articulo esta organizado de la siguiente manera : en la seccion 1 damos el sistema de logica temporal que utilizamos y une breve explicacion del fundamento de nuestras reglas ; en la seccion 2 las reglas que definen el método de resolucion ; la seccion 3 explica de manera suscinta el método de los tableaux semanticos comparandolo con el método de resolucion y, finalmente, en la seccion 4 damos las conclusiones.

1 - LOGICA TEMPORAL

El sistema de lógica temporal que utilizamos es el sistema de lógica temporal lineal definido por Manna y Pnueli [Ma - Pn] . Este sistema supone los siguientes axiomas y reglas de inferencia :

A_1 - $\Vdash$ $\Box$ $(A \supset B) \supset (\Box A \supset \Box B)$

A_2 - $\Vdash$ $O(A \supset B) \supset (OA \supset OB)$

A_3 - $\Vdash$ $O \neg A \equiv \neg O A$

A_4 - $\Vdash$ $\Box A \supset A \,\&\, O A \,\&\, O \Box A$

A_5 - $\Vdash$ $\neg \Box A \equiv \Diamond \neg A$

A_6 - $\Vdash$ $\Box(A \supset OA) \supset (A \supset \Box A)$

A_7 - $\Vdash$ A Until B $\equiv$ B $\vee$ (A & O(A Until B))

A_8 - $\Vdash$ A Until B $\supset \Diamond$ B

Règle R_1 : si A es una tautologia entonces $\Vdash$ A
Règle R_2 : si $\Vdash$ A y $\Vdash$ A $\supset$ B entonces $\Vdash$ B (Modus Ponens)
Règle R_3 : si $\Vdash$ A alors $\Vdash$ $\Box$ A (generalización)

Previo a la definición de las nuevas reglas, consideremos la formulacion del principio de resolución para la lógica clásica :

"Dadas dos cláusulas C_1 y C_2, si existe un literal L_1 en C_1 que es complementario de un literal L_2 en C_2, entonces eliminamos L_1 y L_2 de C_1 y C_2 respectivamente, y construimos la disyuncion de las clausulas restantes".

Por ejemplo, si $C_1 = \neg p \vee C'_1$ y $C_2 = p \vee C'_2$, aplicando el principio de resolucion obtenemos

$$\frac{\neg p \vee C'_1 \qquad p \vee C'_2}{C'_1 \vee C'_2}$$

Podemos considerar que el principio de resolucion se basa en la inferencia tautologica

$$\frac{\neg C'_2 \supset p \qquad p \supset C'_1}{\neg C'_2 \supset C'_1}$$

y que se formula de la primer manera para aplicarlo a formulas en forma normal.

Ahora bien si consideramos formulas temporales de tipo $\Box p$, $\Diamond q$, $0\ p$, p Until q, donde p, q, son literales y la definicion $\Box A = \neg \Diamond \neg A$, donde A es una formula temporal, podemos definir las siguientes reglas de resolucion:

1) $$\frac{\Box p \vee C'_1 \qquad \Diamond \neg p \vee C'_2}{C'_1 \vee C'_2}$$

que estara basada en la inferencia tautologica

$$\frac{\neg C'_1 \supset \Box p \qquad \Box p \supset C'_2}{\neg C'_1 \supset C'_2}$$

2) $$\frac{\Box p \vee C'_1 \qquad \Box \neg p \vee C'_2}{C'_1 \vee C'_2}$$

basada en la inferencia tautologica

$$\frac{\neg C'_1 \supset \Box p \qquad \Diamond p \supset C'_2}{\neg C'_1 \vee C'_2}$$

Suponemos aqui el axioma $\Box A \supset \Diamond A$.

3) $$\frac{\Box p \vee C'_1 \qquad \neg p \vee C'_2}{C'_1 \vee C'_2}$$

basado en la inferencia tautologica

$$\frac{\neg C'_1 \supset \Box p \qquad p \supset C'_2}{\neg C'_1 \supset C'_2}$$

Suponemos aqui el axioma $\Box A \supset A$.

4) $$\frac{\Box p \vee C'_1 \qquad 0\ \neg p \vee C'_2}{C'_1 \vee C'_2}$$

basada en la inferencia tautologica

$$\frac{\neg C'_1 \supset \Box p \qquad 0\ p \supset C'_2}{\neg C'_1 \supset C'_2}$$

Suponemos aqui el axioma $\Box p \supset 0\ p$ y $0\ \neg A \equiv \neg\ 0\ A$.

5) $$\frac{\Diamond A \vee C'_1 \qquad \Diamond B \vee C'_2}{\Diamond (A \,\&\, \Diamond B) \vee \Diamond (B \,\&\, \Diamond A) \vee C'_1 \vee C'_2}$$

donde A, B poseen la forma de una conjuncion de clausulas.

Esta regla se basa en la formula valida:

$\Diamond A \,\&\, \Diamond B \supset \Diamond(A \,\&\, \Diamond B) \vee \Diamond(B \,\&\, \Diamond A)$, que podemos llamar "axioma de linearidad".

En el caso del operador Until, tendremos reglas de tipo

$$\frac{p \text{ Until } q \vee C'_1 \qquad \Box \neg q \vee C'_2}{C'_1 \vee C'_2}$$

basada en la inferencia tautologica:

$$\frac{\neg C'_1 \supset p \text{ Until } q \qquad \Diamond q \supset C'_2}{\neg C'_1 \supset C'_2}$$

Suponemos aqui el axioma $p \text{ Until } q \supset \Diamond q$.

$$\frac{p \text{ Until FALSO}}{\text{FALSO}}$$

La definicion de nuestro método exige también la utilizacion de reglas de transformacion de formulas. Estas reglas se basaran en equivalencias entre formulas de la logica temporal.

Las reglas seran

$$\frac{\Box A}{\Box\Box A} \qquad \text{y}$$

$$\frac{\Diamond A}{A \vee \Diamond(\neg A \,\&\, 0\, A)}$$

lo cual nos permitira aplicar las reglas de resolucion a clausulas que expresan la induccion.

Por ejemplo, si tenemos el conjunto de clausulas insatisfacible

p
$\Box(\neg p \vee 0\, p)$
$\Diamond \neg p$

Las clausulas p y $\Box(\neg p \vee 0\, p)$ expresan una induccion, ya que $(p \,\&\, \Box(p \supset 0\, p)) \supset \Box p$. El conjunto de las tres clausulas es inconsistente pero no podemos obtener una refutacion aplicando las reglas mencionadas anteriormente. Para obtener una refutacion, vamos a definir la transformacion del operador $\Diamond$ de manera mas general:

$$\Diamond A = A \vee 0\, A \vee \ldots \vee 0^{n-1} A \vee \Diamond(\neg A \,\&\, 0 \neg A \,\&\, \ldots \,\&\, 0^{n-1} \neg A \,\&\, 0^n A)$$

donde el valor de n esta dado por el numero de operadores 0 (siguiente) que aparecen en la clausula que expresa la induccion.

El método que vamos a definir es sintactico en el sentido de que no existe referencia a la nocion de modelo. Es por esta razon que debemos definir la nocion de forma normal para las formulas de la logica temporal.

Mas aún, los elementos de una forma normal seran cerrados en relacio a las reglas de resolución. En efecto, podemos considerar las reglas de resolución como operaciones sobre cláusulas, es decir, que si consideramos una formula F en forma normal conjunctiva

$F = \&_{i=1}^{m} C^i$, donde cada C^i es una cláusula

y R una regla de n argumentos, entonces

$$R(C_1, \ldots, C_n) = C$$

donde C debe ser una cláusula.

La forma normal sera definida como una extensión a los operadores temporales de la forma normal conjunctiva clásica. Asi, una fórmula F estará en forma normal conjunctiva si ella posee la forma de una conjunción de cláusulas donde cada cláusula es una disjunción, ya sea de literales, o de literales precedidos por une secuencia finita de operadores O, o de expresiones de la forma □ A(donde A posee la forma de una cláusula) ; o de expresiones de la forma ◇ A (donde A posee la forma general de una conjunción de cláusulas) ; o de expresiones de la forma A Until B, donde A posee la forma general de una cláusula y B la forma general de une conjunción de cláusulas.

Tenemos un teorema de la forma normal que establece que para toda formula F de la logica temporal existe una formula F', en forma normal conjunctiva, equivalente a F. [Ca]

Previamente a la definicion del metodo, damos un procedimiento de reducción, que constituye en el mismo un método de decision para la lógica temporal. [Ca]

Este procedimiento esta basado en el trabajo de Carnap [Car], y reduce el problema de decision para un conjunto de formulas en forma normal, al problema de decision para un conjunto de formulas en forma normal, que se obtienen a partir de las primeras, pero que poseen un número mas pequeno de operadores temporales.

Este procedimiento de reducción es téoricamente correcto pero ineficaz desde el punto de vista de la automatización. Los operadores temporales son olvidados en el procedimiento de reducción y no se puede saber cual era el conjunto de partida. Se necesitan reglas que memoricen las reducciones realizadas.

Estas reglas serán definidas de una manera analoga a la de la resolución clásica en el sentido siguiente :

el objetivo de la resolución clásica es el de eliminar dos literales complementarios de dos cláusulas dadas, por analogia, el objetivo de la resolución temporal sera el de eliminar dos elementos complementarios en cada etapa de una refutación.

En lógica clásica, basta con aplicar una sola regla, la regla Σ_1 (p, ¬ p), pero en logica temporal sera necesario aplicar más de una regla para poder eliminar literales complementarios y poder obtener la cláusula vacia, en el caso de que el conjunto sea insatisfacible.

La expresión $\Sigma_i(C_1, C_2)$ significa que aplicamos la regla Σ_i a las cláusulas C_1 y C_2. Si eliminamos C_1 y C_2, por aplicación de la regla Σ_i, esto significa que el conjunto $\{C_1, C_2\}$ es inconsistente. En el mismo sentido, la expresión $\Gamma_i(C_1)$ significa que aplicamos las reglas Γ_i a subformulas de C_1. Si, como resultado de la aplicación de las reglas eliminamos C_1, esto significa que C_1 es inconsistente.

Nuestro método de resolución tendra la forma de un conjunto de reglas de tipo $\Sigma_i(A,B)$, $\Gamma_j(C)$, donde A, B y C son cláusulas. Las reglas serán definidas de manera recursiva.

En la siguiente sección, damos las nuevas reglas de resolución que definen el método de resolución para la logica temporal lineal.

2. METODO DE RESOLUCION PARA LA LOGICA TEMPORAL LINEAL

Sean C_1 y C_2 dos cláusulas. Definimos recursivamente reglas de la forma

$\Sigma_i((C_1, C_2)$ $\quad i = 1,\ldots,6$

$\Gamma_j(C_1)$ $\quad j = 1,2$

de la siguiente manera :

2.1 Reglas clásicas

a) $\Sigma_1(p, \neg p)$

$\emptyset$ ($\emptyset$ indica la clausula vacia)

- p será denominado un literal resuelto

b) $\Sigma_1((D_1 \vee D_2, F)$

$\Sigma_i(D_1, F) \vee D_2$

para $i \in \{1,\ldots,6\}$. En general, se chequea para que valor(es) de i se aplica la regla.

En este caso $D_1 \vee D_2$ no es una clausula unitaria, pero esta regla se aplicara a subformulas.

La regla puede ser aplicada de manera simétrica, es decir, podemos aplicarle a D_2 y F, obteniendo la expresion $\Sigma_i(D_2, F) \vee D_1$.

c) $\Gamma_1(D_1 \,\&\, D_2 \,\&\, F) = \Sigma_i(D_1, D_2) \,\&\, F)$

2.2 Reglas Temporales

Reglas del operador $\Box$

a) Sea $C_1 = \Box E$ y $C_2 = \Delta F$ entonces definimos

$\Sigma_2(\Box E, \Delta F)$

$\Delta\Sigma_i(E, F)$ donde Δ es $\Box$, $\Diamond$ o 0, para $i \in \{1,\ldots,6\}$.

b) Sea $C_1 = \Box E$ y $C_2 = F$, definimos

$$\frac{\Sigma_2(\Box E, F)}{\Sigma_i(E,F) \qquad \text{para } i \in \{1,\ldots,6\}}$$

Reglas del operador $\Diamond$

c) Sea $C_1 = \Diamond E$ y $C_2 = \Diamond F$ definimos

$$\frac{\Sigma_3 (\Diamond E, \Diamond F)}{\Sigma_i (E, \Diamond F) \vee \Sigma_j (\Diamond E, F) \quad \text{para } i,j \in \{1,\ldots,6\}}$$

d) Sea $C_1 = \Diamond (D_1 \,\&\, D_2 \,\&\, F)$ definimos

$$\frac{\Diamond (D_1 \,\&\, D_2 \,\&\, F)}{\Diamond (\Sigma_i(D_1, D_2) \,\&\, F) \qquad \text{para } i \in \{1,\ldots,6\}}$$

Reglas del operador 0

e) Sea $C_1 = 0\, E$ y $C_2 = O\, F$ definimos

$$\frac{\Sigma_4 (0\, E, 0\, F)}{0\, \Sigma_i (E, F) \qquad \text{para } i \in \{1,\ldots,6\}}$$

Reglas del operador Until

f) Sea $C_1 = E$ Until D y $C_2 = F$, definimos

$$\frac{\Sigma_5 (E \text{ Until } D, F)}{E \text{ Until } \Sigma_i(D, F)}$$

g) Sea $C_1 = E$ Until D y $C_2 = F$, definimos

$$\frac{\Sigma_5 (E \text{ Until } D, F)}{\Sigma_i(E,F) \text{ Until } D}$$

2.3 Reglas de Transformacion

a) $$\frac{\Sigma_6 (\Box E , F)}{\Sigma_i(\Box \Box E, F) \qquad \text{para } i \in \{1,\ldots,6\}}$$

b) $$\frac{\Sigma_6 (\langle\rangle E, F)}{\Diamond_n E \vee \Sigma_i (\Diamond (\Box_{n-1} \neg E \wedge O^n E), F) \quad \text{para } i \in \{1,6\}}$$

donde

$$\Diamond_n E = E \vee O E \vee \ldots \vee O^{n-1} E$$

$$\Box_{n-1} \neg E = E \wedge O E \wedge \ldots \wedge O^{n-1} E$$

y para un conjunto de clausulas S, $n \leq k$, donde k es el numero de operadores 0 que aparecen en S.

2.4 Clausulas Resolubles

Definicion. Sean C_1 y C_2 dos clausulas, diremos que C_1 y C_2 son *resolubles* (respectivamente, C_1 es *resoluble*) si podemos aplicar una de las reglas Σ_i, i= 1,...,6 (respectivamente una de las reglas Γ_i, i=1,2).

2.5 Reglas de simplificacion

Definicion. Sean C_1 y C_2 dos clausulas resolubles (respectivamente C_1 una clausula resoluble), entonces una clausula $R(C_1, C_2)$ (respectivamente $R(C_1)$) se denomina una *resolvente* de C_1 y C_2 (respectivamente de C_1) si ella es el resultado de sustituir:

Ø	por	toda	ocurrencia	de (Ø & E)
E	"	"	"	(Ø v E)
Ø	"	"	"	Δ Ø, donde Δ es $\Box$, $\Diamond$ o 0^j, j>=0;
E	"	"	"	Ø Until E
Φ	"	"	"	E Until Ø

en Σ_i (C_1, C_2), (Γ_j (C_1)), $i \in \{1,...,6\}$, $j \in \{1,2\}$, tantas veces como sea posible.

2.6 Etapas de la resolucion

2.6.1 Sean C_1 v C y C_2 v C' dos clausulas. La regla de inferencia

i) $$\frac{C_1 \text{ v } C \qquad C_2 \text{ v } C'}{R(C_1, C_2) \text{ v } C \text{ v } C'}$$

sera aplicada si C_1 y C_2 son resolubles.

ii) $$\frac{C_1 \text{ v } C}{R(C_1) \text{ v } C}$$

sera aplicada si C_1 es resoluble.

2.6.2 Sea E(DvDvF) una clausula. La siguiente regla sera aplicada

iii) $$\frac{E(D \text{ v } D \text{ v } F)}{E(D \text{ v } F)}$$

donde, en general , E(E') indica que E' es una subformula de E.

2.7 Deduccion y Refutacion

Definicion. Sea S un conjunto de clausulas. Una *deduccion* de C a partir de S es una secuencia finita de clausulas $C_1,...,C_n$, tales que:
-C_n es C
-C_i ($1 \leq i \leq n$) es :
. una clausula de S o,
. una clausula obtenida a partir de C_j, $j < i$ utilizando las reglas de inferencia ii) o iii) o
. una clausula obtenida a partir de C_j y C_k $j,k < i$, utilizando la regla de inferencia i).

Una deduccion de la clausula vacia se llama una *refutacion*.

El método de resolucion para la logica temporal lineal es completo para el sistema axiomatico definido por los axiomas y reglas de los operadores □, ◇, O y es "sound" para el sistema que incluye los axiomas del operador Until.

El teorema de completitud posee la formulacion siguiente : un cojunto S de clausulas es insatisfacible si y solamente si S es refutable.

3. LOS TABLEAUX SEMANTICOS

El método de los tableaux semanticos para la logica temporal se basa en el método de los tableaux para la logica clasica [Be]. En este trabajo no daremos los detalles de dicho método pero en sintesis podemos decir que este método procede de la manera siguiente : descompone una formula en sus componentes elementales, es decir, en formulas atomicas (una variable proposicional o su negacion) o en aquéllos que poseen O como conectivo principal.

Esta descomposicion permite separar las exigencias formuladas por la formula en exigencias sobre el estado inicial (formulas atomicas) y en exigencias sobre el resto de la secuencia (las formulas que poseen O como conectivo principal).

A partir del procedimiento de descomposicion, se aplica un procedimiento de construccion de un grafo modelo. Este grafo representa todos los modelos potenciales de la formula y del mismo se podra extraer un modelo de la formula.

Una vez construido el grafo se realizara un procedimiento de chequeo y eliminacion para determinar si todos los modos del grafo son satisfacibles y si todas las eventualidades pueden ser realizadas.

Una formula sera satisfacible si el procedimiento de chequeo y eliminacion, suponiendo cada vez que una sola proposicion atomica es verdadera, no produce el grafo vacio.

3.1 En lo que sigue, vamos a comparar el método de los tableaux semanticos y el método de résolucion que hemos definido en 2.

Los ejemplos han sido elegidos tomando como punto de partida los dados por Wolper [Wo].

Ejemplo : Daremos como ejemplo un conjunto de clausulas satisfacible, porque si partimos de un conjunto insatisfacible, el procedimiento de descomposicion nos daria el conjunto vacio. Partimos del conjunto $S = \{\Box (q \supset \neg (p \text{ Until } r)), \Box q\}$, aplicamos el procedimiento de descomposicion suponiendo que la formula atomica q es verdadera.

Los pasos en el procedimiento de descomposicion seran los siguientes

1 - $\Sigma q = \{\{\Box (q \supset (\neg p \text{ Until } r)), \Box q\}$
2 - $\Sigma q = \{\{q \supset (\neg p \text{ Until } r), O \Box (q \supset (\neg p \text{ Until } r)), \Box (q \supset (\neg p \text{ Until } r))^* q, O \Box q, \Box q ^*\}$

Simplificacion por q.

3 - Σq = {{r, (($\neg$ p Until r)*, O $\Box$ (q $\supset$ ($\neg$ p Until r)),
$\Box$ (q $\supset$ ($\neg$ p Until r))*, O $\Box$ q, $\Box$ q *},
{$\neg$ p $\wedge$ O ($\neg$ p Until r))*, ($\neg$ p Until r)*, O $\Box$ (q $\supset$ ($\neg$ p Until r)),
$\Box$ (q $\supset$ ($\neg$ p Until r))*
O $\Box$ q, $\Box$ q *}}

por simplificacion por q.

4 - {$\neg$ p $\wedge$ O($\neg$ p Until r)*, ($\neg$ p Until r)*, O $\Box$ (q $\supset$ ($\neg$ p Until r),
$\Box$ (q $\supset$ ($\neg$ p Until r))*, O $\Box$ q, $\Box$ q *}}

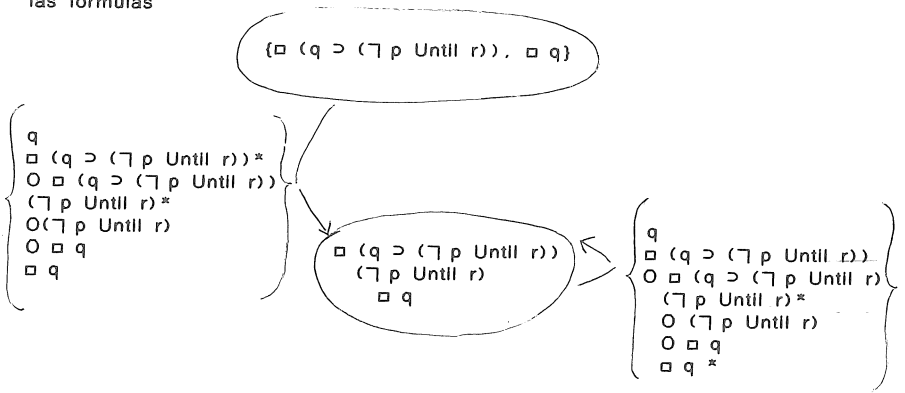
5 - {O ($\neg$ p Until r), ($\neg$ p Until r)*, O $\Box$ (q $\supset$ ($\neg$ p Until r)),
$\Box$ (q $\supset$ ($\neg$ p Until r))*, O $\Box$ q, $\Box$ q *}}

El procedimiento de descomposicion suponiendo la proposicion atomica p verdadera, se detiene en el segundo paso, dado que en el conjunto que se obtiene la proposicion q sera falsa, lo que hara que el conjunto sea insatisfacible. Lo mismo sucede cuando suponemos r verdadera :

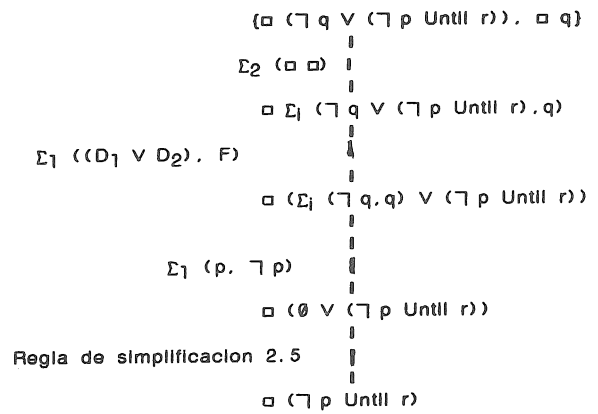
1 - Σp = {{$\Box$(q $\supset$ ($\neg$ p Until r)), $\Box$ q}}

2 - Σp = {q $\supset$ ($\neg$ p Until r)), O $\Box$ (q $\supset$ ($\neg$ p Until r)), $\Box$ (q $\supset$ ($\neg$ p Until r))*
q, o $\Box$ q}}

Una vez realizado el procedimiento de descomposicion, se construye el grafo modelo, a partir del cual se obtendran todos los modelos posibles de las formulas



Si aplicamos el método de resolucion definido en la seccion 2, el arbol de resolucion sera el siguiente, previa transformacion de la formula $\Box$ (q $\supset$ ($\neg$ p Until r)) en forma de clausula :



4. CONCLUSION

La aplicacion del método de resolucion aparece como mas simple, sobre todo cuando se trata de decidir la satisfabilidad de formulas donde aparecen eventualidades, es decir, formulas gobernadas por ◇ o por Until, dado que p Until q ⊃ ◇ q.

El método de los tableaux semanticos permite obtener un modelo de la formula : cada secuencia que es un modelo sera un camino en el grafo y cada camino finito que se obtiene a partir del grafo es el prefijo de algun modelo.

Pero, como lo senalan los mismos autores [Ma – Wo], no todos los caminos infinitos satisfacen a las formulas. Sobre todo cuando se trata de formulas con eventualidades, estas pueden no ser verificadas. Para resolver este problema, se podrian elegir los caminos del grafo que satisfacen las eventualidades, pero esto llevaria a obtener un programa que poseera como ejecuciones posibles solamente algunas de las secuencias que satisfacen las especificaciones. A su vez, esto lleva a los procesos involucrados a interactuar siguiendo un esquema prefijado.

Por ejemplo, si las especificaciones son

$$\Box \Diamond a \wedge \Box \Diamond b$$

el camino elegido podria producir la secuencia a, b, a, b, a, b... como modelo. Lo que es correcto pero inaceptable en situaciones donde a, por ej., puede repetirse mucho mas rapido que b.

Obtener un modelo en el cual se pueda decidir de manera dinamica como resolver las eventualidades solo se puede lograr con ciertas hipotesis de equidad y restricciones sobre las formulas.

Se define un criterio que chequea que en todos los caminos infinitos, o bien la eventualidadad se realiza, o bien, infinitamente a menudo, es "posible", lo que bajo la hipotesis de equidad, garantiza que la eventualidad sera realizada.

El método de resolucion aplicando transformaciones sintacticas sobre las formulas, demuestra que las especificaciones son consistentes con respecto a ciertas propriedades, particularmente, a propiedades donde estan involucradas eventualidades.

La ventaja de nuestro método con respecto al método de los tableaux sémanticos, es que no necesitamos construir un modelo para probar que las especificaciones verifican las propriedades deseadas.

Esta caracteristica sera de primera importancia cuando se trate de demostrar propiedades de vivacidad de un sistema, incluyendo la evolucion, no bloqueo, individual y global, etc.

La parte del método que corresponde a las reglas de los operadores □, ◇, O ha sido implementada bajo la forma de un demonstrador PROTEMP, en el laboratorio Lenguajes y Sistemas, de la Univ. Paul Sabatier, Toulouse.

Bibliografia

[Be] Beth E.W., The foundations of Mathematics, North Holland 1959.

[Ca] Cavalli A.R., Une méthode de décision pour la Logique Temporelle Linéaire. Application à la spécification et vérification des protocoles, Thèse de Doctorat d'Etat, Université de PARIS VII, Diciembre 1984.

[Ca-Fa] Cavalli A.R. y Farinas del Cerro L., A decision mezthod for linear temporal logic, Proceedings of the 7th International Conference on Automated Deduction, Napa, California, USA, Springer Verlag No 170, May 1984.

[Car] Carnap R., Modalities and quantification, JSL Vol 11, 1946, pp33-64.

[La-Owl] Lamport L. and Owicki S., Proving liveness properties of concurrent programs, ACM Transactions on programming languages and systems, Vol. 4, No 3, July 1982.

[Ma-Pn] Manna Z. y Pnueli A., Verification of concurrent programs: the temporal framework. In the Correcteness Problem in Computer Science, Academic Press New York, 1981, pp 215-273.

[Ma-Wo] Manna Z. y Wolper P., Synthesis of Communicating Processes from Temporal Logic Specifications, ACM Transactions on Programming, Languages and Systems, Vol 6, No 1, January 1984, pp 68-93.

[Ro] Robinson J., A machine oriented logic based on the resolution principle, J. ACM, 12, 1965, 23-41.

[Wo] Wolper P., Specification and synthesis of communicating processes using an extended temporal logic, Proceedings of the 9th ACM Symposium on Principles of Programming Languages (Albuquerque, N.M. Jan 25-27), ACM, New York, 1982, pp 20-33.

CONCEPTION ET MISE EN ŒUVRE D'UN SYSTEME DE PREUVE POUR UN SOUS ENSEMBLE DE CCS SANS TRANSMISSION DE VALEURS

A. SORIANO * - J. VOIRON **

RESUME

Nous présentons les principes et la réalisation d'un système de preuve pour un sous ensemble du calcul CCS (Calculus of Communicating Systems), permettant de comparer deux descriptions d'un système communiquant. Nous considérons successivement les cas des comportements finis et des comportements infinis réguliers, sans transmission de valeur. Le système est réalisé en PROLOG-C sur VAX.

ABSTRACT

We present the principles and the realization of a proof system for a subset of the Calculus of Communicating Systems CCS. We consider finite terms and infinite terms without value passing. This system permets the comparison of two system descriptions. The system is implemented in PROLOG-C on VAX.

* Licenciado en Computación (U.C.V. 1977); sémantique, spécification, preuve, construction de programmes; systèmes communicants; intelligence artificielle. Universidad Central de Venezuela, actuellement dans le Groupe Spécification et Analyse des Systèmes Informatiques, Laboratoire Génie Informatique, IMAG .

** Maître Assistant. Université de Grenoble. Thèse 1976; compilation, sémantique, spécification, preuve, construction de programmes; systèmes communicants. Groupe Spécification et Analyse de Systèmes Informatiques. Laboratoire de Génie Informatique, IMAG.

LGI - IMAG BP 68 38410 St. Martin d'Hères Cedex France

Les comportements asynchrones de systèmes communicants peuvent être décrits d'une manière élégante comme termes d'un calcul. On peut alors définir dans ce calcul différentes notions d'équivalence entre comportements. Nous proposons la définition d'un système de preuve basé sur ce principe.

Nous considérons ici le calcul CCS (Calculus of Communicating Systems) proposé par R. Milner [Mi 80]. Nous nous intéressons à deux classes particulières de comportements:

- les comportements finis,
- les comportements infinis de systèmes de transitions décrits par des définitions récursives rationnelles.

Le système développé permet de prouver l'équivalence de deux comportements représentés par des expressions CCS sans transmission de valeurs; dans une démarche de conception descendante, on peut comparer la description d'un système communiquant à sa spécification. Nous étudions les cas suivants:

- Comportements finis (termes sans variables de comportement), pour lesquels nous pouvons décider si deux termes donnés sont ou non *observationnellement congrus* au sens de Milner [Mi 80].
- Comportements infinis rationnels particuliers: correspondant à des systèmes de transitions étiquetées dont l'ensemble d'états est fini. Cette restriction est réaliste dans beaucoup d'applications pratiques, par exemple: protocoles; pour ceci nous imposons certaines restrictions syntaxiques aux expressions de comportement.
  Dans ce cas, nous pouvons décider par bisimulation, si deux termes donnés sont ou non *fortement congrus* au sens Milner [Mi 80] et on peut par suite, approcher la *congruence observationnelle*, en utilisant par exemple, des techniques analogues à celles proposées par Bergstra et Klop [BK 84], (à paraitre dans un prochain rapport).

Le système est en cours de réalisation, la programmation est faite en PROLOG-C (version 1.5) [CM 81][Pe 84] sur un système UNIX 4.2 supporté par un VAX.

Nous présentons dans la suite un rappel les principaux résultats et notations de CCS, le cas des termes finis, le cas des termes infinis rationnels, et une conclusion.

1.- CCS: Calcul asynchrone des systèmes communicants.

Nous rappelons les éléments de CCS et définissons les notations utilisées dans la suite.

CCS est construit à partir:

- d'une algèbre permettant de décrire le comportement de processus parallèles,

- d'un ensemble de règles d'inférence à partir duquel est défini la semantique des opérateurs,
- et des relations d'équivalence et de congruence qui nous permettront de comparer les expressions de comportement.

La présentation de CCS faite dans la suite est purement algébrique et les relations d'équivalence et de congruence sont induites par un ensemble d'axiomes.

Algèbre pour CCS.

Soient:

- D, l'ensemble de noms *d'actions ou ports de communication* dont les éléments sont désignés par des lettres minuscules ne commençant pas par la lettre "n",
- ND, l'ensemble de noms de ports, appelés *complémentaires*, dont les éléments sont désignés par des lettres minuscules commençant par la lettre "n",
- la bijection:
  a (appartenant à D) <---> na (appartenant à ND)
  on dit que "na" est le *port complémentaire* de "a",
  ceci nous permet de définir l'interaction entre deux processus,
- t, une étiquette représentant une *communication non observable*, et "t" n'appartient ni à D ni à ND,
- P, l'ensemble d'étiquettes ou ports de communication, P = D U ND,
- F, l'ensemble de fonctions de redéfinition f telles que:
  f: L⊆P ----> M⊆P
  f est une bijection,
  f respecte les compléments définis par la bijection,
- B, l'ensemble des expressions CCS.

Soit la signature: $\Sigma = (P \cup \{t\}, nil, ., +, \|, [], \backslash)$
où:

P U {t} est l'ensemble de noms d'actions, opérateurs d'arité zéro,
nil est la constante pour les comportements,
{.,+,||,[],\} est l'ensemble des opérateurs binaires, avec:

. : P U {t} ----> B
+ : B x B ----> B
|| : B x B ----> B
[] : B x F ----> B
\ : B x P ----> B

La sémantique de ces opérateurs est définie opérationnellement en suivant les idées de Plotkin [Pl 81]:

(.): $a.E \xrightarrow{a} E$ (ACTION)

(+):

$$(1)\quad \frac{E_1 \xrightarrow{a} E'_1}{E_1 + E_2 \xrightarrow{a} E'_1}$$

(CHOIX NON DETERMINISTE)

$$(2)\quad \frac{E_2 \xrightarrow{a} E'_2}{E_1 + E_2 \xrightarrow{a} E'_2}$$

(||)

$$(1)\quad \frac{E_1 \xrightarrow{a} E'_1}{E_1 \,||\, E_2 \xrightarrow{a} E'_1 \,||\, E_2}$$

$$(2)\quad \frac{E_2 \xrightarrow{a} E'_2}{E_1 \,||\, E_2 \xrightarrow{a} E_1 \,||\, E'_2}$$

(COMPOSITION PARALLELE)

$$(3)\quad \frac{E_1 \xrightarrow{a} E'_1 \,,\; E_2 \xrightarrow{\overline{a}} E'_2}{E_1 \,||\, E_2 \xrightarrow{t} E'_1 \,||\, E'_2}$$

(\)

$$\frac{E \xrightarrow{a} E'}{E \backslash b \xrightarrow{a} E'}$$

(EFFACEMENT)

([])

$$\frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}$$

(REDEFINITION)

Relations d'équivalence et de congruence.

Soit l'ensemble d'axiomes suivant:

(+):

- (A_1): $X + (Y + Z) = (X + Y) + Z$
- (A_2): $X + Y = Y + X$
- (A_3): $X + X = X$
- (A_4): $X + nil = X$

(||): (A_5): Si U et V sont de la forme:

$U = \Sigma\ q_i\ .\ X_i$ et $V = \Sigma\ p_j\ .\ Y_j$

alors:

$$U \| V = \Sigma\ q_i\ .\ (X_i \| V) + \Sigma\ p_j\ .\ (U \| Y_j) + \Sigma\ t\ .(X_i \| Y_j)$$
$$q_i = np_j$$
$$\vee\ nq_i = p_j$$

(A_6): $X \| Y = Y \| X$

(A_7): $X \| (Y \| Z) = (X \| Y) \| Z$

(A_8): $X \| nil = X$ $(A_{8'})$: $nil \| X = X$

([]): (A_9): $(p.X)[f] = f(p).(X[f])$

(A_{10}): $(X + Y)[f] = X[f] + Y[f]$

(A_{11}): $nil[f] = nil$

(A_{12}): $X[I] = X$ (I: L —> L fonction identité)

(A_{13}): $X[f] = X[f']$ (X:L et f|`L = f'|`L)

(A_{14}): $X[f][f'] = X[f' \circ f]$

(A_{15}): $(X\|Y)[f] = X[f]\|Y[f]$

(\): (A_{16}): $X[f]\backslash f(q) = X\backslash q[f]$

(A_{17}): $(p.X)\backslash q = \begin{cases} nil & \text{si } p = q \text{ ou } p = nq \\ p.(X\backslash q) & \text{sinon} \end{cases}$

(A_{18}): $(X+Y) = X\backslash q + Y\backslash q$

(A_{19}): $nil\backslash q = nil$

(A_{20}): $X\backslash q = X$ (X:L, $q, nq \notin L$)

(A_{21}): $X\backslash q_1\backslash q_2 = X\backslash q_1\backslash q_2$

(A_{22}): $(X\|Y)\backslash q = X\backslash q \| Y\backslash q$ ($X:L_1$, $Y:L_2$, $q, nq \notin L1 \cap L2$)

(t): (A_{23}): $X+t.X = t.X$

(A_{24}): $p.(X+t.Y) = p.(X+t.Y)+p.Y$

(A_{25}): $p.t.X = p.X$

(A_{26}): $X+t.(X+Y)=t.(X+Y)$

(A_{27}): $t.B = B$

Les différentes relations d'équivalence et de congruence utilisées dans la suite sont:

- l'*équivalence observationnelle* induite par les axiomes (A_1)-(A_{27}) est notée $\approx$,
- la *congruence forte* induite par les axiomes (A_1)-(A_{22}) est notée $\sim$,
- la *congruence observationnelle* induite par les axiomes (A_1)-(A_{26}) est notée $\approx^C$,
- la congruence induite les axiomes (A_1)-(A_5), $(A_{8'})$, (A_8)-(A_{11}), (A_{17})-(A_{19}), (A_{23})-(A_{25}) notée $\approx^C$, est appelée *congruence* .
- la congruence induite par les axiomes (A_1)-(A_2) notée $\approx^S$, et appelée *congruence* (A_1)-(A_2).

Définitions équationnelles de comportements.

Il est possible de définir des comportements au moyen d'équations de la forme:

var <=== B_{var}

où: var est le nom d'une variable (de comportement),
B est une expression de comportement,
<=== est une relation de congruence,

de cette manière, il est possible de définir des comportements infinis.

Nous utilisons dans la suite les définitions suivantes:

On dit qu'une *occurrence d'une variable V est non gardée* dans une expression de comportement ssi elle n'est pas précédée par un nom d'action.

On dit qu'une *variable est non gardée* dans une expression de comportement ssi il existe au moins une occurrence non gardée de la variable dans l'expression.

Par exemple, X est non gardée dans les expressions suivantes:

X , a.Y + X , a.Y + X + b.Z , a.X + X

On dit qu'une *expression de comportement est bien gardée* ssi elle ne contient pas de variables non gardées, ou s'il est toujours possible de substituer chaque occurrence d'une variable par une expression dont les termes ne sont pas de variables non gardées.

Par exemple, X n'est pas une expression de comportement bien gardée dans le système d'equations:

X <=== Y + a.X
Y <=== X + b.nil

par contre X est bien gardée dans:

X <=== Y +a.X
Y <=== b.X +b.nil

et aussi dans: X <=== a.(X + b.nil).

Dans le cas des termes ne comportant pas de variables, nous décidons si deux expressions données sont ou non observationnellement congrues [Mi 80] en transformant chaque terme en une forme normale; intuitivement, la forme normale, d'une expression de comportement CCS sans transmission de valeurs, définie pour la relation de congruence observationnelle, est une expression CCS ne contenant que:

i) des opérateurs - et +,
ii) et le "minimum" de termes nécessaires à l'expression de ce comportement;

chacune de ces formes normales est le représentant d'une classe de la congruence observationnelle. Nous vérifions si ces formes canoniques correspondent ou non à la même classe de congruence, c'est-à-dire si elles sont équivalentes via la relation de congruence(A1)-(A2), notée $\approx^S$

I.- FORME NORMALE.

Pour définir la forme normale d'une expression CCS, nous définissons premièrement la notation d'*expression (écrite) sous la forme d'une somme* et nous prouvons que toute expression décrivant un comportement fini est congrue à une autre expression CCS écrite sous la forme d'une somme; puis nous donnons la définition de *comportements dérivés* d'une expression afin de pouvoir simplifier des termes d'une expression sous la forme d'une somme et finalement pouvoir caractériser la forme normale.

Définition: *expression sous la forme d'une somme.*

(i) nil est sous la forme d'une somme,
(ii) pour tout g appartenant à P U {t}, g.B est sous la forme d'une somme ssi B est sous la forme d'une somme,
(iii) B1+B2 est sous la forme d'une somme ssi B1 et B2 sont sous la forme d'une somme,
(iv) toute expression sous la forme d'une somme est obtenue à partir de (i), (ii) et (iii).

Exemples:

Les expressions suivantes sont sous la forme d'une somme:
a.b.nil+b.d.nil , a.t.(a.nil+t.(b.nil+nd.nil)) , nil

Les expressions suivantes ne sont pas sous la forme d'une somme:
d.b.nil || na.nb.nil , a.b.nil[a/b] , a.b.nil\b

Proposition 1.

Toute expression finie CCS est congrue à une expression écrite sous la forme d'une somme.

Intuitivement, si p est une action appartenant à P U {t}, les p-dérivés d'une expression correspondent aux comportements possibles lorsqu'il y a eu une p-communication.

Définition:

Pour tout p appartenant à P U {t}, le *p-dérivé* d'une expression écrite sous la forme d'une somme $E = \Sigma\, p_i.e_i$, est défini comme suit:

(i) e_i est p_i-dérivé de $\Sigma\, p_i.e_i$,

(ii) tout t-dérivé de e_i est un p-dérivé de $\Sigma\, p_i.e_i$,

(iii) si $p_i = t$ alors tout p-dérivé de e_i est un p-dérivé de $\Sigma\, p_i.e_i$.

Par exemple, soit l'expression: t.(a.nil+t.(b.nil+c.(e.nil+t.(d.nil+t.nil))))

les t-dérivés sont:

a.nil+t.(b.nil+c.(e.nil+t.(d.nil+t.nil)))

b.nil+c.(e.nil+t.(d.nil+t.nil))

les c-dérivés sont:

e.nil+t.(d.nil+t.nil)

d.nil+t.nil

nil

Proposition 3.

Si e' est congru à un p-dérivé de e, alors $p.e' + e \simeq^{C} e$.

En reprenant les idées de Hennesy et Milner [HM 80] on remarque qu'une forme normale ne peut être réduite par simplification; pour garantir que l'axiome a.t.X = a.X n'est pas applicable, nous définissons d'abord la notion de forme normale propre, puis celle de forme normale impropre et en fin celle de forme normale.

Definitions:

On dit qu'une expression sous la forme d'une somme $e = \Sigma\, p_i.e_i$ est une *forme normale propre* ssi:

i) elle n'est pas de la forme t.e,

ii) chaque e_i est sous une forme normale propre,

iii) pour i différent de j, il n'existe pas de p_i-dérivé h de $p_j.e_j$ tel que $e_i \simeq^{S} h$.

On dit qu'une expression sous la forme d'une somme e = t.e' , est une *forme normale impropre* ssi e' est une forme normale propre.

On dit qu'une expression sous la forme d'une somme est une *forme normale* ssi elle est une forme normale propre ou impropre.

Exemples:

Expression	Forme normale
a.nil +a.(b.nil+t.nil)	a.(b.nil+t.nil)
a.nil +t.(b.nil+t.(d.nil+a.nil))+d.nil	t.(b.nil+t(d.nil+a.nil))
a.nil+t.(nil+t.(nil+t.a.nil))	t.a.nil
(a.nil||na.nil)+t.a.na.nil	t.a.na.nil+na.a.nil+t.nil

Nous pouvons énoncer trois thèorémes dont la preuve complete est fourni dans le rapport [SV 85].

Théorème d'existence: Tout terme fini CCS est congru à une forme normale fn.

Théorème d'unicité: Pour tout f, g termes finis CCS sous forme normale, $f \underline{\approx}^{C} g$ ssi $f \approx^{S} g$.

Théorème: Pour tout f,g terme fini CCS, $f \approx^{C} g$ ssi $f \underline{\approx}^{C} g$.

Donc, la congruence observationnelle définie par R. Milner [Mi 80] est exactement la congruence utilisée dans la définition de la forme normale.

2.- ALGORITHME POUR CALCULER LA FORME NORMALE D'UNE EXPRESSION CCS.

L'algorithme qui calcule la forme normale associée à une expression CCS procède en deux étapes:

1.- Calcul de l'expression sous la forme d'une somme, associée à une expression CCS,
2.- Calcul de la forme normale associée à l'expression sous la forme d'une somme calculée.

Algorithme de transformation d'une expression sous la forme d'une somme.

Cet algorithme procède en trois étapes:

1.- Construction de l'arbre d'opérateurs associé à l'expression donnée, cet arbre tient compte de la priorité usuelle pour les opérateurs
   - les feuilles correspondent
     soit à des opérateurs CCS d'arité 0,
     soit à un ensemble de ports de communication,
     soit à une fonction de redéfinition.
   - les autres nœuds correspondent aux opérateurs binaires CCS.

2.- Transformation de l'arbre pour éliminer des opérateurs [],\ et les feuilles nil non necessaires, en utilisant les axiomes: (A_2), (A_4), (A_6), (A_8),

3.- Construction de l'arbre associé à l'expression écrite sous la forme d'une somme, en appliquant récursivement sur l'arbre réduit en partant des feuilles, la définition d'une expression sous forme d'une somme, et la propriété suivante pour éliminer l'opérateur de composition parallèle.

Propriété: Si A et B sont deux expressions CCS sous la forme d'une somme, alors, en appliquant les axiomes (A_5),(A_6) et (A_8) un nombre fini de fois, nous obtenons une expression sous la forme d'une somme observationnellement congrue à A||B .

Algorithme pour construire la forme normale associée à une expression sous la forme d'une somme.

L'algorithme qui calcule la forme normale associée à une expression sous la forme d'une somme procède en deux étapes:
- Construction de l'arbre d'opérateurs sans des 't' précédés par des noms d'actions, par application de l'axiome A_{25},
- Construction de l'arbre associé à la forme normale, en utilisant la définition d'une expression sous forme normale et la proposition 3.

Nous appliquons la définition d'une expression sous forme normale d'une manière récursive sur chaque sous-arbre, en partant des feuilles.

3.- MISE EN OEUVRE.

La réalisation a été faite en C-PROLOG (version 1.5) [Pe 84], [CM 81] sur un système UNIX 4.2 suporté par un VAX; le programme est composé d'une soixantaine de clauses PROLOG; pour avoir une idée de ses limites sur le système considéré, nous donnons le tableau suivant:

EXPRESSIONS CCS	PLACE OCUPEE PAR LES TERMES CONSTRUITS PENDANT L'EXECUTION [mots]	TEMPS MIS POUR DECIDER LA CONGRUENCE OBSERV. ENTRE LES EXPRESSIONS [sec]
t.a.(t.nil+nil)+a.nil	moins de 1188	0.15
t.(nil+t.(nil+t.a.nil))		
t.(a.nil || na.nil)	3628	0.38
t.(na.nil || a.nil)		

Les performances du programme sont évaluées en terme de nombre d'inférences logiques.

III.- TERMES INFINIS.

Nous nous restreignons dans ce cas à des expressions pouvant contenir des variables de comportement $X_1, X_2, \ldots, X_n$ définies comme suit:

$$X_1 <=== S_1(X_1, X_2, \ldots, X_n)$$
$$X_2 <=== S_2(X_1, X_2, \ldots, X_n)$$
$$\ldots$$
$$X_n <=== S_n(X_1, X_2, \ldots, X_n)$$

où $S_1(X_1, \ldots, X_n)$, $S_2(X_1, \ldots, X_n)$, $\ldots$, $S_n(X_1, \ldots, X_n)$ sont des expressions de comportement écrites sous la forme d'une somme.

L'axiomatisation (A_1)-(A_{27}) fournie pour CCS n'est complète que dans le cas de comportements finis. La méthode de preuve décrite dans la partie II ne peut être appliquée pour les comportements infinis.

Dans ce cas, une méthode consiste à construire pour chaque expression un système de transitions $S_E(E)$ (sous forme d'un graphe fini) dont le comportement est sémantiquement identique à celui décrit par l'expression E. Nous pouvons comparer par bisimulation les systèmes $S_{E1}(E1)$, $S_{E2}(E2)$, i=1, 2 ainsi construits, ceci nous permet de décider la congruence forte des termes E_1, E_2 initiaux. Par transformation sur ces systèmes de transitions, nous pouvons décider la congruence observationnelle des termes initiaux par une technique analogue, ce problème n'est pas étudié ici.

1. - CONSTRUCTION D'UN SYSTEME DE TRANSITIONS ETIQUETEES A PARTIR D'UNE EXPRESSION CCS.

Définition [Ke 76]:

Un *système de transitions étiquetées* est un triplet:

$$S_A = (Q, A, \{\xrightarrow{a}\}_{a \in A} \cup \{\xrightarrow{t}\})$$

où: Q est un ensemble d'états,
A est un vocabulaire de noms d'actions,
$\{\xrightarrow{a}\}_{a \in A} \cup \{\xrightarrow{t}\}$ est un ensemble de relations binaires sur Q.

On définit pour chaque système de transitions étiquetées un état initial $Eo \in Q$.

Soit une expression CCS bien gardée:

$$E(X1, \ldots, Xn)$$

où: $X_1 <=== S_1(X_1, X_2, \ldots, X_n)$
$X_2 <=== S_2(X_1, X_2, \ldots, X_n)$
$\ldots$
$X_n <=== S_n(X_1, X_2, \ldots, X_n)$

les variables de comportement sont définies par les expressions $S1(X1, \ldots Xn)$, $\ldots$, $Sn(X1, \ldots, Xn)$ écrites sous forme d'une somme.

Nous associons un système de transitions étiquetées S(E) à l'expression E, comme suit:

$S(E) = (Q_E(E), A_E(E), ST_E(E))$

où: $Q_E(E) = \{ B_1 : \exists B, e \ (B \xrightarrow{e} B_1 \in ST_E(E))\} \cup \{E\}$,

$A_E(E) = \{ e : \exists B, B_1 (B \xrightarrow{e} B_1 \in ST_E(E))\}$,

$ST_E(E)$ est un ensemble de relations binaires sur Q correspondant aux dérivations de l'expression E.

Pour construire cet ensemble, nous procèdons par induction sur la structure de E :

$ST_{E0}(nil) = \emptyset$,

$ST_{E0}(a.E_i) = \{E_0 \xrightarrow{a} E_i\} \cup ST_{Ei}(E_i)$,

$ST_{E0}(E1+E2) = ST_{E0}(E1) + ST_{E0}(E2)$,

$ST_{E0}(E_i[f]) = \{E[f] \xrightarrow{f(e)} E_i[f] : E$ non identique à $E0 \wedge$

$E \xrightarrow{e} E_i \in ST_{E0}(E_i)\} \cup$

$\{Eo \xrightarrow{f(e)} E_i[f] : Eo \xrightarrow{e} E_i \in ST_{E0}(E_i)\}$,

$ST_{E0}(x) = STS_{E0}(B, \emptyset)$, où: x est une variable de comportement définie par:

x<=== B

où $STS_{E0}(B,S)$ est défini par:

si B est identique à $Eo \wedge \exists E_1, e : Eo \xrightarrow{e} E1 \in S$

alors $STS_{E0}(B,S) = S$

sinon $STS_{E0}(nil,S) = S$,

$STS_{E0}(p.B,S) = STS_B(B, S \cup \{Eo \xrightarrow{p} B\})$,

$STS_{E0}(B1+B2,S) = STS_{E0}(B1, STS_{E0}(B2,S))$,

$STS_{E0}(x,S) = STS_{E0}(B,S)$, si x est une variable et x<===B,

$ST_{E0}(E1 \parallel E2) = STP_{E0}(ST_{E1}(E1), ST_{E2}(E2), E1, E2, \emptyset, \emptyset)$

où STP_{E0} (S1,S2,B1,B2,F,S) est l'ensemble de transitions correspondant au système de transitions associé à l'expression E1 || E2. Cet ensemble est obtenu par l'application de l'axiome (A_5) sur les systèmes de transitions associés aux expressions E1 et E2 (définition détaillée dans [SV 85]).

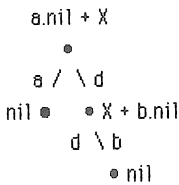
$ST_{E0}(E1 \backslash P) = STE_{E0}(E1 \backslash P)$

En définissant $STE_{E0}(E)$ comme le système de transitions étiquetées associé à l'expression E et en prenant Eo comme le nom de l'état initial (définition detaillée dans [SV 85]).

Exemple:

Soit l'expression CCS: a.nil +X
où: X<---d.(X + b.nil)

le système de transitions associé peut se représenter graphiquement comme:



2.- BISIMULATION.

Intuitivement, une bisimulation est une relation R entre deux systèmes de transitions S1 et S2, R définie entre les états de S1 et S2 comme suit:

A chaque état de S1 on associe au moins un état de S2 à partir duquel on a les mêmes transitions possibles et vice-versa.

Définition [Pa 81a]:

Soient: - deux systèmes de transitions Si = (Qi,Ri), i=1,2 ,
- une relation R de Q1 sur Q2, totale pour Q1 et Q2,

une *bisimulation* entre deux systèmes S1 et S2 (notée $R1 <\overset{R}{===}> R2$) est une relation R telle que:

$q_1 R q_2$ implique $(\forall q'_1 (q_1 R_1 q'_1$ implique $\exists q'_2 (q_2 R_2 q'_2$ et $q'_1 R q'_2))$
$\wedge \forall q'_2 (q_2 R_2 q'_2$ implique $\exists q'_1 (q_1 R_1 q'_1$ et $q'_1 R q'_2)))$.

Pour pouvoir décider s'il existe une relation de bisimulation entre deux systèmes de transitions, nous utilisons le resultat suivant dû à Park [Pa 81b]:
Soient B l'ensemble de comportements,
F, G : B x B ---> B définies comme:

$G(R) = \{(p,q) :$ si $p \xrightarrow{a} p'$ alors $\exists q' (q \xrightarrow{a} q'$ et $p' R q')\}$

$F(R) = G(R) \cap \overline{G(\overline{R})}$, où $\overline{R} = \{(p,q) : p R q\}$

F est monotone et possede un plus grand point fixe:

$F = \cup \{ R$ tels que $R \subseteq F(R)\}$.

Une extension de ce résultat aux systèmes de transitions étiquetées est:
Soient : S1 = (Q1,A1,R1) et S2 = (Q2,A2,R2) deux systèmes de transitions

$R1 <\overset{R}{===}> R2$ ssi les trois conditions suivantes sont vérifiées:

(Condition 1): il existe R tel que R est le plus grand point fixe de $R = R \cap F(R)$,

où: $F(R) = \cap \{ (q1,q2) : (\forall q'_1\ (q_1\ R_{1a} q'_1$ implique $\exists q'_2 (q_2\ R_{2a}\ q'_2 \wedge\ q'_1\ R\ q'_2)))$
$a \in A_1 \cup A_2$
$\wedge\ (\forall\ q'_2\ (q_2\ R_{2a} q'_2$ implique $\exists q'_1 (q_1\ R_{1a}\ q'_1 \wedge q'_1\ R\ q'_2))))$,

(Condition 2): $(EO_1, EO_2) \in R$, où: EO_1 et EO_2 sont les états initiaux de S1 et S2,

(Condition 3): R est totale pour Q1 et Q2.

3.- ALGORITHME POUR PROUVER LA CONGRUENCE FORTE ENTRE DEUX EXPRESSIONS CCS.

L'algorithme de preuve de la congruence forte entre deux expressions CCS procède en trois étapes:

1.- construction des systèmes de transitions étiquetées associés aux expressions CCS données,
2.- calcul itératif d'une relation R entre les systèmes de transition construits, vérifiant la condition 1,
3.- vérification des conditions 2 et 3 pour la relation R.

Algorithme de construction des systèmes de transitions.

Pour la construction de chaque système de transitions:

1) nous construisons d'abord des arbres d'opérateurs correspondant:
   - à l'expression de comportement,
   - à chaque expression de comportement définissant une variable de comportement dans le cas d'un système équationnel.

   Soit Ao l'arbre d'opérateurs associé à une expression CCS avec variables :
   - les feuilles de cet arbre correspondent
     soit à des opérateurs CCS d'arité zéro,
     soit à un ensemble de ports,
     soit à une fonction de redéfinition,
     soit à une variable de comportement.
   - les autres nœuds de l'arbre correspondent aux opérateurs binaires CCS.

   Cet arbre tient compte de la priorité définie en CCS pour les opérateurs.

2) Pour le système de transitions, nous construisons de manière inductive l'ensemble de relations de transition définies précédemment, en partant des feuilles de l'arbre d'opérateurs.

Algorithme de calcul d'une relation R entre deux systèmes de transitions.

L'algorithme de construction d'une relation de bisimulation R entre deux systèmes de transitions $S_1 = (Q_1,A_1,R_1)$ et $S_2 = (Q_2,A_2,R_2)$ calcule itérativement le plus grand point fixe de:

$R = R \cap F(R)$,

où:

$F(R) = \cap \{(q1,q2): (\forall q'_1 \ (q_1 \ R_{1a} q'_1 \ \text{implique} \ \exists q'_2 (q_2 \ R_{2a} \ q'_2 \ \wedge \ q'_1 \ R \ q'_2))) \ a \in A_1 \cup A_2 \ \wedge \ (\forall q'_2 \ (q_2 \ R_{2a} q'_2 \ \text{implique} \ \exists q'_1 (q_1 \ R_{1a} \ q'_1 \wedge q'_1 \ R \ q'_2)))\}$,

en partant de la relation $R = Q_1 \times Q_2$.

Existence d'une relation de bisimulation.

Si la relation calculée R vérifie les conditions 2 et 3 alors c'est une relation de bisimulation.

L'existence d'une bisimulation entre les systèmes de transitions permet de décider si les expressions CCS données sont fortement congrues.

4.- MISE EN ŒUVRE.

La réalisation a éte faite en C-PROLOG (version 1.5) [Pe 84], [CM 81] sur un système UNIX supporté par un VAX; le programme est composé d'environ cent cinquante clauses PROLOG; pour avoir une idée de sa performance, nous donnons l'exemple suivant:

Soit le contrôleur défini dans [Mi 80]:

```
- (s||p||q)\[e,f]
        s<---ne.nil
        p<---e.na.nb.nf.p
        q<---f.nc.nd.ne.q
```

et sa spécification:

```
- x
  x<---t.a.nb.t.c.nd.x
```

la description du contrôleur est correcte (fortement congrue à l'expression de sa spécification); la place occupée par les termes construits pendant l'execution de la preuve est de 151 924 mots, le temps d'execution est de 32.98 secondes.

IV.- CONCLUSIONS.

Un système complet de preuve peut être construit en étendant les rèsultats précédents aux expressions CCS avec transmission de valeur.

D'autres calculs de systèmes communicants peuvent être pris en compte.

[BK 84] Bergstra, J.A., Klop, J.W.
A COMPLETE INFERENCE SYSTEM FOR REGULAR PROCESSES WITH SILENT MOVES.
Centre for Mathematics and Computer Science. Departement of Computer Science. Report CS R8420. November 1984.

[CM 81] Clocksin, W.F, Mellish, C.S.
PROGRAMMING IN PROLOG.
Springer-Verlag. 1981.

[HM 80] Hennessy, M., Milner, R.
Unpublished note.

[Ke 72] Keller, R. M.
VECTOR REPLACEMENT SYSTEMS: A FORMALISM FOR MODELING ASYNCHRONOUS SYSTEMS,
Princeton University, Technical Report No. 117.1972.

[Mi 80] Milner, R.
A CALCULUS FOR COMMUNICATING SYSTEMS.
Lecture Notes in Computer Science, 92. Springer-Verlag. 1980.

[Pa 81a] Park, D.
CONCURRENCY AND AUTOMATE ON INFINITE SEQUENCES.
Proc. 5th. G. I. Conference. Lecture Notes in Computer Science, 104. Springer-Verlag. 1981.

[Pa 81b] Park, D.
A NEW EQUIVALENCE NOTION FOR COMMUNICATING SYSTEMS.
Bulletin of European Association for Theoretical Computer Science, 14, pp. 78-80. 1981.

[Pe 84] Pereira, F.
C-PROLOG USER'MANUAL.
SRI International, Menlo Park, California. 1984.

[Pl 81] Plotkin, G. D.
A STRUCTURAL APPROACH TO OPERATIONAL SEMANTICS.
Aarthus University, Computer Science Departement, Technical report DAIMI FN-19. 1981.

[Si 84] Sifakis, J.
PROPERTY PRESERVING HOMOMORPHISMES OF TRANSITIONS SYSTEMS.
Workshop Carnegie Mellon, University Pittsburgh, P.A. 1983. Lecture Notes in Computer Science, 164, pp 458-473. Springer-Verlag. 1984.

[SV 85] Soriano, A., Voiron, J.
CONCEPTION ET MISE EN ŒUVRE D'UN SYSTEME DE PREUVE POUR UN SOUS ENSEMBLE DE CCS SANS TRANSMISSION DE VALEURS.
IMAG. Rapport de Recherche N° 530. 1985

UM ESBOÇO DE UMA LÓGICA DE RESOLUÇÃO DE PROBLEMAS

M.A.LOPES* e B.M.ACIOLY**

SUMÁRIO

Neste trabalho é apresentado um esboço teórico de uma lógica de esquema de soluções funcionais e também um esboço de uma lógica de esquema de problemas. A ligação entre os dois sistemas formais (lógicas) é feita através da noção de satisfação apresentada aqui, através dos conceitos de realização e solubilidade. A partir da noção da satisfação é dado um esboço para uma lógica de resolução de problemas.

ABSTRACT

This paper presents a framework of a funcional solution scheme logic and a framework of a problem scheme logic. The connection of these two formal systems (logics) is done through the notion of satisfability, which in turn is presented using the concepts of realization and solubility. Finally, from the notion of satisfability a framework of a problem solving logic is derived.

* Bacharel em Matemática (UFPE, 1969), Mestre em Matemática (UFPE,1971), Doutor em Informática (PUC/RJ,1981), teoria de problemas, inteligência artificial, tipos abstratos de dados, Sociedade Brasileira de Computação, Rua Ramiro Costa, 79 - Prado - Recife, 50.000 - (081) 228.2922

** Bacharel em Matemática (UFPE,1970), Mestre em Matemática (UFPE,1980) lógica, teoria de problemas, inteligência artificial, Rua Gal. Polidoro, 425/201 - Iputinga - Recife, 50.000 - (081) 271.2740.

1. INTRODUÇÃO

Uma lógica de problemas, como sugere Kolmogoroff em [1], com sua interpretação para o cálculo proposicional intuicionista, leva em conta fortemente a noção de construção de solução, sem necessariamente, se referir ao problema em si. Em seu trabalho, Kolmogoroff apresentou a sintaxe da lógica intuicionista e dá a ela uma semântica de "problema" sem levar em conta a natureza estrutural de problemas. Considerando um problema como uma estrutura relacional, de modo como faz Lopes em [2], não é possível encontrar o similar à interpretação dada por Kolmogoroff para $\neg P$, que significa: "se dada uma solução para o problema P, obtenha uma contradição". Isto porque, em [2], um problema é um construto, onde sua negação, do ponto de vista da solubilidade, não é definível univocamente, uma vez que envolve simultaneamente a noção de intencionalidade e de admissibilidade, enquanto em [1] a noção de problema é pragmática, sem uma estruturação prévia. O aproveitamento da sintaxe da lógica intuicionista para uma interpretação de problemas, no sentido de Lopes, parece inadequado porque a interpretação da negação está prejudicada como veremos na secção 3.1.

A idéia básica deste trabalho é tratar de problemas e solução como entidades sintaticamente distintas e consequentemente de lógicas (sistemas formais) distintas. Desse modo, uma lógica de solução tratará de soluções como esquemas (funcionais) abstratos, que estarão posteriormente sujeitos a realizações, ou seja, dado um esquema de solução e um problema, esse esquema é realizável nesse problema se ele pode ser interpretado como uma certa função do conjunto de dados no conjunto de resultados desse problema.

A ligação entre os dois sistemas lógicos será feita através de uma noção de satisfação (não tarskiana) onde um esquema a satisfaz um problema P se e somente se a realizado em P implica em a resolve P.

2. PROBLEMAS

A palavra *problema* é empregada correntemente em várias acepções. No nosso caso, estamos interessados nas situações em que podemos distinguir claramente uma intenção de associar situações iniciais e situações finais, ou em outras palavras, em todo problema distinguimos três elementos importantes: o *domínio de dados*, um *domínio de resultados*, que representam a parte substantiva do problema, e um *predicado intencional* que representa a intenção (subjetiva). Numa questão, como por exemplo: "calcular o comprimento de listas" identificamos: o conjunto L de listas como o domínio de dados, os conjuntos $\mathbb{N}$ dos naturais como o domínio de resultados e o predicado intencional r expresso do seguinte modo: para todo elemento $\ell \in L$ e para todo $n \in \mathbb{N}$, ℓ está relacionado com n por meio de r se e somente se, o comprimento de ℓ for igual a n.

Embora os três elementos listados acima sejam fundamentais em qualquer problema, eles não são os únicos a aparecerem. No exemplo dado, poderíamos está interessados somente em listas de tamanho finito, ou seja, impomos uma condição sobre os dados, ou poderíamos só admitir soluções algorítmicas. Podemos, portanto, dar a seguinte definição

de problemas, veja Lopes [2].

Def. 2.1. - Um *problema* é uma estrutura do tipo $P = \langle D,R,d,r,A\rangle$, onde D e R são conjuntos não vazios, chamados *domínios de dados* e de *resultados*, respectivamente, $d \subseteq D$, representa a *condição sobre os dados*, $r \subseteq D \times R$ (produto cartesiano de D por R) representa a *condição do problema* e $A \subseteq R^D$ (conjunto das funções de D em R), representa a *condição de admissibilidade*.

Os conjuntos D e R representam a parte substantiva (concreta), ou seja, os universos do problema, enquanto d, r e A representam a parte intencional (subjetiva). A idéia é que r reflita a condição principal ou intencionalidade principal, segundo a qual, os elementos $f \in D$, $s \in R$ se relacionam, isto é, $(f,s) \in r$; d reflita a parte de D relevante no problema e, A seja a restrição as muitas possíveis maneiras de mapear D em R (funções), que satisfazem r e que serão consideradas admissíveis.

Def. 2.2. - Um problema $P = \langle D,R,d,r,A\rangle$ é *solúvel* se existe uma função $a \in R^D$ tal que $a \in A$ e para todo $f \in d$ tem-se $(f,a(f)) \in r$. Uma função que satisfaça estas condições chama-se *solução* do problema P e usamos a notação $a \mid\!\!-\!\!- P$ para expressar que "a resolve P" ou "a é uma solução de P".

A partir dos problemas $P = \langle D_1,R_1,d_1,r_1,A_1\rangle$ e $Q = \langle D_2,R_2,d_2,r_2,A_2\rangle$, podemos construir novos problemas como seguem. Ver [4]

$P \wedge Q = \langle D_2 \times D_2, R_1 \times R_2, d_1 \times d_2, r_1 \times r_2, A_1 \times A_2\rangle$, onde "×", indica o produto cartesiano, e lemos o "problema P e Q".

$P \vee Q = \langle D_1 \times D_2, R_1 \times R_2, d_1 \times d_2, r_1 \cup r_2, A_1 \times A_2\rangle$, onde "×" é o produto cartesiano $R_1 \cup r_2 = \{(x,y)/x \in r_1$ ou $y \in r_2\}$ (união cartesiana) PvQ será lido "P *ou* Q"

$P \longrightarrow Q = \langle D_1 \times D_2, R_1 \times R_2, d_1 \times d_2, \overline{r}_1 \cup r_2, A_1 \times A_2\rangle$, onde $\overline{r}_1$, é o complementar de r_1 em relação a $D_1 \times D_1$. O problema $P \longrightarrow Q$ será lido: "P implica Q".

Como podemos ter aprioristocamente soluções funcionais, isto é, esquemas abstratos que poderão ser interpretados como funções através de atribuições de entradas (argumentos) e saídas (valor da função) dentro de domínios definidos, devemos ter um mecanismo formal que permita essa atribuição. No caso de funcionais e problemas isso será feito com a seguinte definição:

Def. 2.4. - Uma solução funcional (ou simplesmente funcional) é *realizável* num problema $P = \langle D,R,d,r,A\rangle$ se a interpretação a^P de a em P é uma função $a^P : D \longrightarrow R$. Quando não houver motivo de confusão a^P será simplesmente denotado por a. A notação $a \mid\!\!\longrightarrow P$ será utilizada para expressar a é realizável em P.

Def. 2.5 - Um funcional a *satisfaz* um problema P (denotado por: $a \mid\!\!\longrightarrow P$) se e so

mente se, $a \longmapsto P$ implica $a \vdash\!\!\!\!- P$, isto é $a \in R^D$ implica $a \in A$ e $\forall f \in d(f, a(f)) \in r$.

A definição 2.5 será utilizada mais tarde para fazer a ligação entre os sistemas formais de soluções funcionais e esquema de problemas.

3. LÓGICA DE PROBLEMAS

3.1. A QUESTÃO DA NEGAÇÃO

É natural que a idéia de negação de um problema seja obtida através da negação de uma intenção, isto é, se $P = \langle D,R,d,r,A \rangle$, então $\neg P = \langle D,R,d,\bar{r},A \rangle$ onde $\bar{r}$ é a complementação (com relação a $D \times R$) da intenção r. Isso significa que r e $\bar{r}$ são disjuntas ou em outras palavras, independentes, no sentido que as intenções descritas em r e $\bar{r}$ são independentes. Por exemplo: Dada uma sequência de números naturais, sem repetição, ordene-a! Claramente a intenção no problema acima é a ordenação. A negação desta intenção pode ser dada por "não ordene" ou "desordene". Não ordene parece ser a alternativa lógica da negativa que consistiria em acescentar o conectivo "não" antes da ação "ordene". Por outro lado, isso não chega a ser uma ação em si, uma vez que, "não aja", não tem valor de procedimentos. Desse modo, "desordene" seria uma alternativa para a negação da intenção do problema acima.

Considerando "desordene" como negativa da intenção acima, poderemos apontar o seguinte inconveniente técnico: se descrevermos ordene através da conjunção de dois predicados "*mesma*" e "*boa*", onde *mesma* (s,s') significa que as sequências s e s' possuem os mesmos elementos e *boa* (s') significa que s' está ordenada, temos que ordene (s,s') (ou seja, s' é a sequência s ordenada) é equivalente a *mesma* $(s,s') \wedge boa(s')$. assim "desordene" seria descrita por $\neg$ ordene (s,s') ou seja $\neg((mesma(s,s') \wedge boa(s'))$ que é equivalente a $\neg\, mesma(s,s') \vee \neg\, boa(s')$ (1) que não corresponde a intuição de "desordene". Na realidade a fórmula predicativa que refletiria a intuição de "desordene" seria mesma$(s,s') \wedge \neg\, boa(s')$ (2) que não é equivalente a fórmula acima. Isto significa que para estabelecermos (2) em vez de (1), para expressarmos "desordene" teríamos que fazer considerações extralógica e isto requer uma análise semântica a priori da estrutura sintática da intenção do problema.

Além do acima exposto, a negação de uma ação com valor de procedimento parece adquirir o "*status*" de uma ação em si, independente da ação positiva, isto é, ela também é uma ação positiva. Se encarado assim, os problemas P e $\neg P$ são problemas diferentes e independentes. É claro que a negação de um problema, do ponto de vsita da intenção, não dá a essa negação, o carater de negativa lógica (conectivo lógico).

Pelas considerações acima parece-nos adequado que um sistema formal de problemas sirva de interpretação para o cálculo proposicional positivo clássico, como está em [3].

3.2. SISTEMA FORMAL DE ESQUEMAS DE PROBLEMAS

Seja E o conjunto das expressões de uma linguagem L formadas conforme as regras abaixo.

As letras A,B,C,...,P,Q,...,Z, são variáveis e os símbolos $\wedge$, $\vee$ e $\longrightarrow$, são conectivos lógicos. Na linguagem L as fórmulas bem formadas, aqui chamadas de *esquemas de problemas* ou simplesmente problemas, são obtidas através das seguintes regras:

P_1 - As variáveis A,B,C,... etc são (esquemas de) problemas

P_2 - Se A e B são problemas, então $A \wedge B$, $A \vee B$ e $A \longrightarrow B$ são problemas.

P_3 - Os problemas são as sentenças obtidas através de P_1 e P_2.

A igualdade entre dois problemas será introduzida aqui, pela definição tradicional:

$$A = B \overline{\underset{\text{def.}}{}} (A \longrightarrow B) \wedge (A \longrightarrow B)$$

O *cálculo de problemas* será definido através do seguinte conjunto de esquemas de axiomas e regras de inferência:

A_1 - $A \longrightarrow B \longrightarrow A$

A_2 - $(A \longrightarrow (B \longrightarrow C)) \longrightarrow (A \longrightarrow B) \longrightarrow (A \longrightarrow C)$

A_3 - $((A \longrightarrow B) \longrightarrow A) \longrightarrow A$

A_4 - $A \longrightarrow (A \vee B)$

A_5 - $A \longrightarrow (B \vee A)$

A_6 - $(A \longrightarrow C) \longrightarrow (B \longrightarrow C) \longrightarrow (A \vee B) \longrightarrow A$

A_7 - $A \wedge B \longrightarrow A$

A_8 - $(A \wedge B) \longrightarrow B$

A_9 - $(A \longrightarrow B) \longrightarrow (A \wedge B)$

MP - Se A e $A \longrightarrow B$, então B.

Denotando "é dedutível de" por "$\vdash$", e assumindo as definições clássicas de "prova", "dedutível" e "teorema" temos os seguintes resultados que são consequências diretas dos axiomas e da regra de inferência acima.

1 - Seja Γ um conjunto de problemas e A, B são problemas. Se $\Gamma, A \vdash B$, então $\Gamma \vdash A \longrightarrow B$ (teorema da dedução).

2 - $A \longrightarrow B, B \longrightarrow C \vdash A \longrightarrow C$.

3 - $\vdash A \longrightarrow (A \longrightarrow B) \longrightarrow B$

4 - $\vdash (A \longrightarrow (A \longrightarrow B)) \longrightarrow A \longrightarrow B$

5 - $(A \longrightarrow B) \longrightarrow B, A \longrightarrow C \vdash (C \longrightarrow B) \longrightarrow B$

6 - $\vdash A \vee (A \longrightarrow B)$

7 - $A \vee B, A \longrightarrow C, B \longrightarrow D \vdash C \vee D$

8 - $((A_1 \vee A_2 \vee \ldots \vee A_n) \vee B) = (A_1 \vee \ldots A_n \vee B)$

9 - $A_1 \vee \ldots \vee A_n \vdash B_1 \vee \ldots \vee B_k$ se cada dos $A_1, \ldots, A_n$ é um dos $B_1, \ldots, B_k$.

10 - $A \vee B, C \longrightarrow A \vdash (B \longrightarrow C) \longrightarrow A$.

11 - $A \vee B, A \longrightarrow B \vdash B$

12 - $\vdash (A \longrightarrow (B \wedge C)) = (A \longrightarrow B) \wedge (A \longrightarrow C)$

13 - $\vdash ((A \wedge B) \longrightarrow C) = (A \longrightarrow B \longrightarrow C)$

14 - $\vdash (C \vee (A \wedge B)) = ((C \vee A) \wedge (C \vee B))$

3.3. SISTEMA FORMAL DE SOLUÇÕES FUNCIONAIS

De modo análogo ao Sistema Formal de Esquemas de Problemas, usando as letras minúsculas $a, b, c, \ldots$ como variáveis e os símbolos $\wedge$, $\vee$, $\longrightarrow$ como conectivos lógicos, podemos definir para um Sistema Formal de soluções funcionais, as fórmulas bem formadas, aqui chamadas de *soluções funcionais* ou simplesmente funcionais, como sendo obtidas através de regras similares a da seção 3.2:

S_1 - As variáveis $a, b, c, \ldots$ são (soluções funcionais) funcionais

S_2 - Se a e b são funcionais então $a \wedge b$, $a \vee b$ e $a \longrightarrow b$ são funcionais.

Consideremos, de modo similar ao da seção 3.2, um esquema de axiomas e regras de inferências para o cálculo de soluções funcionais

S_1 - $a \longrightarrow b \longrightarrow a$

S_2 - $a \longrightarrow (b \longrightarrow c) \longrightarrow (a \longrightarrow b) \longrightarrow (a \longrightarrow c)$

S_3 - $((a \longrightarrow b) \longrightarrow a) \longrightarrow a$

S_4 - $a \longrightarrow (a \vee b)$

S_5 - $a \longrightarrow (b \vee a)$

S_6 - $(a \longrightarrow c) \longrightarrow (b \longrightarrow c) \longrightarrow (a \vee b) \longrightarrow a$.

S_7 - $a \wedge b \longrightarrow a$

S_8 - $a \wedge b \longrightarrow b$

S_9 - $(a \longrightarrow b) \longrightarrow (a \wedge b)$

MP Se a e $a \longrightarrow b$, então b.

Como no cálculo de problemas, temos também aqui, resultados similares aqueles, em decorrência dos axiomas e regra de inferência acima.

Devemos observar que um sistema formal de soluções funcionais foi utilizado para interpretar também o cálculo proposicional positivo clássico, uma vez, que a negação de funcionais não tem sentido lógico, isto é, se a é um funcional, $\neg a$ para ter sentido, deva ser interpretado (por ex: $-a$, a^{-1}, etc) e isso tira a condição de conectivo lógico que gostaríamos de dar a negação ($\neg$). Desse modo, nos parece razoável imaginar

a formação de funcionais através das regras P_1, P_2 e P_3, do início desta secção.

4. LÓGICA DE RESOLUÇÃO DE PROBLEMAS

Nesta secção faremos a junção dos dois sistemas formais vistos na secção 3, utilizando para isto, a definição 2.5 de satisfação. Usando expressões do tipo $a \mid\!\longmapsto A$, onde a e A são sentenças da lógica de soluções funcionais e de problemas, respectivamente, e as letras gregas α, β, γ,... para simbolizar as sentenças de um novo sistema formal, vamos construir as *fórmulas bem formadas* para o sistema formal de resolução de problemas do seguinte modo:

1 - As expressões do tipo $a \mid\!\longmapsto A$ e $a \mid\!\not\longmapsto A$ (a não satisfaz o problema A) são fórmulas bem formadas;

2 - Se α e β são fórmulas bem formadas então $\neg\alpha$ $\alpha \overset{\wedge}{\wedge} \beta$, $\alpha \overset{\vee}{\vee} \beta$ e $\alpha \Longrightarrow \beta$ também o são;

3 - As fórmulas bem formados são somente as obtidas pelas regras acima.

O cálculo de resolução de problemas será definida através dos seguintes esquemas de axiomas e regras de inferência:

Ax.1 - $\alpha \Longrightarrow \beta \Longrightarrow \alpha$

Ax.2 - $(\alpha \Longrightarrow (\beta \Longrightarrow \gamma)) \Longrightarrow ((\alpha \Longrightarrow \beta) \Longrightarrow (\alpha \Longrightarrow \gamma))$

Ax.3 - $(\alpha \Longrightarrow \beta) \Longrightarrow (\neg\beta \Longrightarrow \neg\alpha)$

e

Rf.1 - $\dfrac{\alpha,\ \alpha \Longrightarrow \beta}{\beta}$

Rf.2 - $\dfrac{a \mid\!\longmapsto A,\ b \mid\!\longmapsto B}{a \longrightarrow b \mid\!\longmapsto A \longrightarrow B}$ regra implicativa

Rf.3 - $\dfrac{a \mid\!\longmapsto A,\ b \mid\!\longmapsto B}{a \wedge b \mid\!\longmapsto A \wedge B}$ regra conjuntiva

Rf.4 - $\dfrac{a \mid\!\longmapsto A,\ b \mid\!\longmapsto B}{a \vee b \mid\!\longmapsto A \vee B}$ regra disjuntiva

Se observarmos que os esquemas de axiomas acima e a regra Rf.1 são os mesmos para o cálculo proposional clássico, podemos concluir que para as sentenças construídas a partir de sentenças do tipo $a \mid\!\longmapsto A$, são válidos os resultados similares aos daquele cálculo clássico. Por exemplo, uma versão restrita do teorema da dedução como segue:

se

$$\frac{a_1 \mid\!\longmapsto A_1,\ a_2 \mid\!\longmapsto A,\ \ldots, a_n \mid\!\longmapsto A_n;\ a \mid\!\longmapsto A}{b \mid\!\longmapsto B}$$

então

$$\frac{a_1 \mid\!\!\longmapsto A_1,\ a_2 \mid\!\!\longmapsto A_2,\ldots,a_n \mid\!\!\longmapsto A_n}{a \mid\!\!\longmapsto A \Longrightarrow b \mid\!\!\longmapsto B}\ .$$

O resultado que vem a seguir estabele de modo mais completo a ligação, que pretendemos, entre os cálculos de soluções funcionais e problemas.

Teorema 4.1 - Se $a \mid\!\!\longmapsto A$, $b \mid\!\!\longmapsto B$ e $c \mid\!\!\longmapsto C$ então

1 - $a \longrightarrow (b \longrightarrow a) \mid\!\!\longmapsto A \longrightarrow (B \longrightarrow A)$
2 - $a \longrightarrow (b \longrightarrow c) \longrightarrow ((a \longrightarrow b) \longrightarrow (a \longrightarrow c)) \mid\!\!\longmapsto A \longrightarrow (B \longrightarrow C) \longrightarrow ((A \longrightarrow B) \longrightarrow (A \longrightarrow C))$
3 - $((a \longrightarrow b)) \longrightarrow a) \longrightarrow a \mid\!\!\longmapsto ((A \longrightarrow B) \longrightarrow A) \longrightarrow A$
4 - $a \longrightarrow a \vee b \mid\!\!\longmapsto A \longrightarrow A \vee B$
5 - $a \longrightarrow b \vee a \mid\!\!\longmapsto A \longrightarrow B \vee A$
6 - $(a \longrightarrow c) \longrightarrow ((b \longrightarrow c) \longrightarrow (a \vee b \longrightarrow a)) \mid\!\!\longmapsto (A \longrightarrow C) \longrightarrow ((B \longrightarrow C) \longrightarrow (A \vee B \longrightarrow A)$
7 - $a \wedge b \longrightarrow a \mid\!\!\longmapsto A \wedge B \longrightarrow A$
8 - $a \wedge b \longrightarrow a \mid\!\!\longmapsto A \wedge B \longrightarrow B$
9 - $(a \longrightarrow b) \longrightarrow (a \wedge b) \mid\!\!\longmapsto (A \longrightarrow B) \longrightarrow (A \wedge B)$

Faremos a prova do item 6 e deixamos ao leitor, as provas dos demais itens, que podem facilmente ser feitas de modo análogo.

Prova do item 6:

1 - $a \mid\!\!\longmapsto A$
2 - $b \mid\!\!\longmapsto B$ hipóteses
3 - $c \mid\!\!\longmapsto C$
4 - $a \vee b \mid\!\!\longmapsto A \vee B$ usando (1,2 e RF.4)
5 - $a \vee b \longrightarrow a \mid\!\!\longmapsto A \vee B \longrightarrow A$ (4,1 e RF.2)
6 - $b \longrightarrow c \mid\!\!\longmapsto B \longrightarrow C$ (2,3 e RF.2)
7 - $(b \longrightarrow c) \longrightarrow (a\ b \longrightarrow a) \mid\!\!\longmapsto (B \longrightarrow C) \longrightarrow (A \vee B \longrightarrow A)$ (6,5 e RF.2)
8 - $a \longrightarrow c \mid\!\!\longmapsto A \longrightarrow C$ (1,3 e RF.2)
9 - $((a \longrightarrow c) \longrightarrow (b \longrightarrow c) \longrightarrow (a \vee b \longrightarrow a))) \mid\!\!\longmapsto ((A \longrightarrow C) \longrightarrow (B \longrightarrow C) \longrightarrow (A \vee B \longrightarrow A)))$
(8,7 e RF.2) □

Teorema 4.2 - Se $a_1, a_2, \ldots, a_n$ é uma prova de a no cálculo de soluções funcionais e $A_1, A_2, \ldots, A_n$ é, correspondentemente, a prova de A no cálculo de problemas então $a \mid\!\!\longmapsto A$

Prova: Decorre diretamente da definição de prova para o cálculo proposional positivo, o teorema 4.1 e a regra (modus ponens) RF. 1 □

Corolario 4.3 - Se a e A são teoremas dos cálculos de solução funcionais e de problemas então $a \mid\!\!\longmapsto A$.

Prova: óbvia □

Devemos observar que o teorema 4.1 compatibiliza o conceito de satisfação, do

cálculo de resolução de problemas, com as axiomáticas dos cálculos de soluções funcionais e de problemas, no sentido de preservar as estruturas sentenciais similares dos dois cálculos. O Teorema 4.2, decorrente fortemente do Teorema 4.1, expressa que a satisfação também preserva os mecanismos de derivação dos dois cálculos, o que era desejável!

Algumas sentenças do cálculo de resolução de problemas merecem especial atenção. Por exemplo, a sentença $a \mid\!\!\longmapsto A \longrightarrow B$ deve ser entendida como $a \mid\!\!\longmapsto A \Longrightarrow a \mid\!\!\longmapsto B$. Por outro lado, $a \longrightarrow b \mid\!\!\longmapsto A$ deve ser entendida como $a \mid\!\!\longmapsto A \Longrightarrow b \mid\!\!\longmapsto A$. Isso, entretanto, pode, no caso da sentença $a \longrightarrow b \mid\!\!\longmapsto A \longrightarrow B$, gerar sentenças não necessariamente equivalentes, quando o desdobramenro for feito primeiramente pelo lado direito do sinal $\mid\!\!\longmapsto$ (1º caso) ou pelo lado esquerdo (2º caso). Porém, vale o seguinte resultado:

Proposição 4.4 - Se $a \mid\!\!\longmapsto B$ e $b \mid\!\!\longmapsto B$ então são logicamente equivalentes as sentenças abaixo.

1 - $(a \mid\!\!\longmapsto A \Longrightarrow a \mid\!\!\longmapsto B) \Longrightarrow (b \mid\!\!\longmapsto A \Longrightarrow b \mid\!\!\longmapsto B)$
2 - $(a \mid\!\!\longmapsto A \Longrightarrow b \mid\!\!\longmapsto A) \Longrightarrow (a \mid\!\!\longmapsto B \Longrightarrow b \mid\!\!\longmapsto B)$

Prova: fácil □

O significado da proposição acima, é, que debaixo das hipóteses dadas, é indiferente que o desdobramento da sentença $a \longrightarrow b \mid\!\!\longmapsto A \longrightarrow B$ seja feito pela direita ou pela esquerda.

5. CONCLUSÃO

Este esboço tem o objetivo de fornecer indicações da viabilidade técnica do desenvolvimento de uma lógica de resolução de problemas baseada na conceituação de problemas, proposto em [2]. O estabelecimento de um cálculo de resolução de problemas constitui uma etapa imprescindível para o desenvolvimento posterior de máquinas de inferência de solução de problemas. Além do preenchimento das lacunas aqui deixadas e a explicitação de muitos resultados desejáveis para esse cálculo, como por exemplo, uma versão mais geral do teorema de dedução, teoremas de compactação, a extensão para um cálculo de 1ª ordem etc, é natural que seja tentada uma modelação dentro de ambientes contextuais de problemas, ainda a nível de cálculo proposicional.

Gostaríamos de chamar a atenção sobre a idéia da ligação de dois sistemas formais aqui apresentada, sem que para isso tenha sido utilizada a noção de interpretação entre Teorias. Eventualmente, essa idéia poderia ser também usada para ligar outros sistemas formais, por exemplo, aritmética de Zermelo-Fraenkel e a lógica booleana.

REFERÊNCIAS

[1] KOLMOGOROFF, A.N., Zur Deutung der Intuitionistischen Logik, Mathematische Zeitschrift, vol. 35, pp. 58-65, 1932.

[2] LOPES,M.A., Introdução a uma Teoria Geral de Problemas, Tese de Doutorado, Departamento de Informática - PUC/RJ - maio/81.

[3] RASIOWA,H., An Algebraic Approach to Non-classical Logics, North-Holland Publishing Campany, 1974.

[4] LOPES,M.A. & VELOSO,P.A.S., Operations on Problems and their Solution Spaces, Cybernetics and Systems Reseach, R. Trappe (ed.) Noth-Holland Publishing Campany, 1982.

UN LENGUAJE FORMAL DE ESPECIFICACION DE PROCESOS COMUNICANTES PARA EL ANALISIS DE REDES.

Ph. JORRAND*, J. M. PEREIRA**

RESUMEN

Un lenguaje formal para la especificación de Procesos Comunicantes es presentado. Dos modelos operacionales equivalentes le son asociados: un sistema de transición de estados y un sistema de reescritura de términos con eventos.
Sobre este lenguaje se introduce un cálculo para la construcción de redes de procesos. Este cálculo es una manipulación sintáctica de símbolos del lenguaje formal que nos permite reducir el análisis de una red al análisis de un proceso simple.

ABSTRACT

A formal language for the specification of Communicating Processes is presented. Two equivalent operational models are associated with it: a state transition system and a term rewriting system with events.
On top of this language we introduce a calculus for the construction of networks of processes. This calculus is syntactic manipulation of symbolic of the formal language which allows us to reduce the analysis of a network to the analysis of a single process.

* Director de Investigación en el CNRS. Teoría de la Programación, Informatica Fundamental, Especificación, Procesos Comunicantes, IMAG-BP 68-38402
St. Martin d'Hères cedex - FRANCE.

** Ingeniero en Computación. Magister en Ciencias de la Computación, Docteur "Informatique"; Teoría de la Programación-Teoría de Tipos-Semántica de Lenguajes-Sistemas de Procesos; Universidad Simón Bolívar-Dpto. de Matemáticas-AP 80659-CARACAS-VENEZUELA.

** Este trabajo ha sido realizado bajo el patrocinio financiero del CONICIT (VENEZUELA), el Servicio Cultural de la Embajada de Francia en Venezuela y el Ministerio de Relaciones Exteriores de Francia.

Un *"proceso"* puede ser considerado como una caja (máquina, agente) capaz de hacer cálculos y de enviar y recibir mensajes a través ciertos puntos de conexión ligados a la caja. Estos puntos de conexión son llamados *"puertas"* de entrada y/o salida del proceso. Ellos constituyen los únicos medios por los cuales el proceso se puede comunicar con su medio ambiente.

Esta capacidad de transmitir y recibir información hacia o por el medio ambiente permite construir *Sistemas de Procesos* que se comunican (SPC). En estos sistemas los procesos intercambian mensajes entre ellos a través de sus puertas. Así una *"comunicación"* es considerada como la transmisión instantanea de un mensaje desde una puerta de entrada hacia una puerta de salida.

Los Sistemas de Procesos que deseamos modelar son tales que:

- los procesos que conforman el sistema evolucionan de forma independiente, salvo en los momentos en los cuales ellos se comunican. Esto introduce todas las posibilidades del *paralelismo*.
- para que una comunicación tenga lugar ella debe ser sincronizada. Esto quiere decir que la comunicación se realiza solamente en el caso en el proceso que transmite y el proceso que recibe estén listos para intercambiar el mensaje. Esta hipótesis sobre las comunicaciones ha sido introducida por C.A.R. Hoare y se la conoce bajo el nombre de *"rendez-vous"*.
- varias comunicaciones pueden ser realizadas simultaneamente. El modelo describe entonces secuencias de conjuntos de comunicaciones simultáneos. Estos conjuntos serán denominados *"eventos"*.
- la construcción de un sistema de procesos será un caso particular de la especificación de un proceso simple. Es decir un SPC es una frase de la notación sintáctica utilizada para describir un proceso simple.
- podamos estudiar el comportamiento del proceso independiente de su medio ambiente. Para ello, un proceso es considerado como una *"unidad aislada"*. Para asirlarla, nosotros asociamos, a cada puerta, una *"puerta ficticia"* llamada δ que representa cualquier puerta de cualquier medio ambiente. Debido a este aislamiento no se puede hacer ninguna suposición sobre el orden en el cual un medio ambiente se va a comunicar con el proceso. Luego la descripción de un proceso debe considerar todas las secuencias posibles de eventos que él puede realizar i.e. todos los medios ambientes en los cuales él puede ser utilizado.

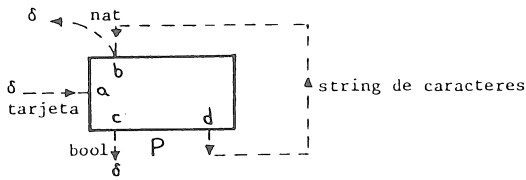
En la sección 2 se introduce el lenguaje formal para describir procesos y dos modelos semánticos para dicho lenguaje formal. Ambos modelos son operacionales y equivalentes.

En la sección 3 se introduce un cálculo formal que nos permite construir los SPC. Este calculo es una manipulación sintáctica de frases del lenguaje formal definido en la sección 2 que permite reducir un SPC a un proceso simple.

Finalmente en la sección 4 se discuten las posibilidades de analizar el comportamiento de un SPC bajo el modelo propuesto y la potencia de este lenguaje.

2 SINTAXIS Y SEMANTICA FORMAL DE UN PROCESO.

La estructura de un proceso puede ser considerada como un diagrama del tipo:



Así, un proceso posee:

-un nombre

-un conjunto Puertas formado por los identificadores de sus puertas

-un conjunto de conexiones entre puertas definidas como un subconjunto de

$K = K_{IN} \cup K_{EX}$

donde

$K_{IN} \subseteq$ Puertas x Puertas	conexiones internas
$K_{EX} \subseteq K_{en} \cup K_{sa}$	conexiones externas
$K_{en} \subseteq \{\delta\}$ x Puertas	conexiones de entrada
$K_{sa} \subseteq$ Puertas x $\{\delta\}$	conexiones de salida

Los elementos $<x,y> \in K$ son denotados por x,y y ellos representan la conexión entre la puerta de salida x y la puerta de entrada y.

Cada conexión en el proceso tiene asociado un identificador de dominio que denota el tipo de mensaje que puede circular por la conexión.

El comportamiento de un proceso expresa las diferentes secuencias de eventos permitidas y las diferentes funciones calculadas. Al nivel del diagrama, esto expresa cuales son las diferentes secuencias en que las conexiones se disparan y cual es el mensaje que ese momento se está transmitiendo.

.1 LA SINTAXIS FORMAL DE UN PROCESO.

Un *lenguaje* L para describir procesos es un par $L = <D,L>$ donde D es un conjunto de *identificadores de dominios*

L consiste en tres firmas:

-un conjunto F de *símbolos de operación* con una función de aridad

$\rho_F : F \to D^* \times D$ donde D^* es el conjunto de secuencias finitas de elementos de D, incluyendo la secuencia vacia λ.

-un conjunto K de *símbolos de conexión* con una función de aridad ρ_K: $K \to D$.

-un conjunto E de *símbolos de relación* con una función de aridad $\rho_E : E \to D^*$

Sea $V(L) = \{V_d \mid d \in D\}$ una familia de conjuntos no-vacios indexada por D denominada *variables* de L.

Sea $D' = D \cup \{estado, evento\}$ t.q. estado, evento $\notin D$

Los *términos* de L, $T(L) = \{T_{d'} \mid d' \in D'\}$ es la mas pequeña familia de conjuntos indexada por D' t.q.

i. $V_d \subseteq T_d$ para todo $d \in D$

ii. si $f \in F$ y $\rho_F(f) = <\lambda, d>$ entonces $f \in T_d$ para todo $d \in D$

iii. si $f \in F$ y $\rho_F(f) = <d_1, \ldots d_n, d>$ y $t_i \in T_{d_i}$ para $1 \leq i \leq n$ entonces $f(t_1, \ldots, t_n) \in T_d$

iv. si $\{k_1, \ldots, k_n\} \subseteq K$ y $\{k_1, \ldots, k_n\} \neq \emptyset$ y $\rho_k(k_i) = d_i$ y $t_i \in T_{d_i}$ para $1 \leq i \leq n$ entonces $\{k_1(t_1), \ldots, k_n(t_n)\} \in T_{evento}$

v. $\tau \in T_{evento}$ (τ denota el evento vacio i.e. $\emptyset$)

vi. si $e \in E$ y $\rho_E(e) = d_1 \ldots d_n$ y $t_i \in T_{d_i}$ para $1 \leq i \leq n$ entonces $e(t_1, \ldots, t_n) \in T_{estado}$.

Los conjuntos T_d engendrados por las reglas i)iii)iii) son denominados *términos de operación* y denotados por T_{oper}.

Para cada $t \in T(L)$ la familia de variables que aparecen en t es denotada por var(t). En el caso que $t \in T_{evento}$, var(t) esta dividida en dos

$$var_{en}(t) = \bigcup_{t_i \in E_n(t)} var(t_i) \quad \text{donde } E_n(t) = \{t' \mid k\ (t') \in t \wedge k \in K_{en}\}$$

$$var_{sa}(t) = \bigcup_{t_i \in S_a(t)} var(t_i) \quad \text{donde } S_a(t) = \{t'/k(t') \in t \wedge k \in K_{so}\} \cup K_{IN}$$

Las *reglas* de L es el conjunto definido por

$$R(L) = \{t_1 \xrightarrow{t_2} t_3 \;/\; t_1, t_3 \in T_{estado} \wedge t_2 \in T_{evento} \wedge var_{sa}(t_2), var(t_3) \subseteq var(t_1) \cup var_{en}(t_2)\}$$

Luego *un proceso* es decrito por una tripleta

$$p = < L, I, R >$$

donde

L es un lenguaje para describir procesos

$I \subseteq T_{\text{estado en } L}$ t.q. $I \neq \emptyset \wedge \forall t \in I \,.\, var(t) = \emptyset \wedge I$ finito

$R \subseteq R(L)$ t.q. R es finito

Por comodidad en los ejemplos se utilizan la siguiente notación para la descripción de un proceso

p = proceso

operaciones $f_{1,1},\ldots,f_{1,n} : d_{1,1}\ldots d_{1,i} \rightarrow d_1$

$\vdots$

conexiones $k_{1,1},\ldots,k_{1,r} : d_i$

$\vdots$

estados $e_{1,1},\ldots,e_{1,p} : d_{i,1}\ldots d_{i,j}$

$\vdots$

iniciales $t_1,\ldots,t_k$

reglas $t_{1,1} \xrightarrow{t_{2,1}} t_{3,1}$

$\vdots$

Ejemplo:

Sea Nat el identificador del dominio de los naturales

Sec(Nat) el identificador de dominios de las secuencias de los naturales

El proceso Pila recibe naturales por su puerta de entrada i y los envia por su puerta de salida o en orden LIFO (último en entrar primero en salir)

Pila = proceso

operaciones cons: Sec(Nat), Nat → Sec(Nat.)

$\wedge$: → Sec(Nat)

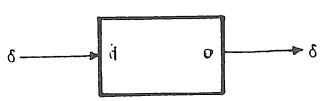
conexiones $\delta \cdot i$, $o.\delta$: Nat

estados P : Sec(Nat)

inicial $P(\wedge)$

reglas $P(w) \xrightarrow{\{\delta \cdot i(n)\}} P(cons(w,n))$

$P(cons(w,n)) \xrightarrow{\{o.\delta(n)\}} P(w)$



Pila

Se proponen en esta sección dos modelos semánticos para las descripciones de procesos y se muestra su equivalencia.

El primer modelo interpreta la descripción de un proceso $p = < L, I, R >$ como un sistema de transición de estados, i.e. un automata, en general no-determinístico e infinito.

El segundo modelo propuesto tambien operacional interpreta la descripción $p = < L, I, R >$ como un sistema de transición de términos con Eventos, i.e. una especie de sistema de Reglas de Reescritura con Eventos incorporadas a las reglas.

Otros modelos semánticos han sido asociados a la descripción de procesos [Per 84]: la presentación de una variedad de álgebras y un árbol en el espacio ultramétrico de los árboles infinitos. Estos modelos permiten estudiar ciertas propiedades del comportamiento de un SPC que serán objeto de futuras publicaciones.

2.2.1 LA DESCRIPCION DE UN PROCESO: UN SISTEMA DE TRANSICION DE ESTADOS.

Un sistema de transición de estados está definido como una tupla $S=(Q,T,Rel)$ donde:

- Q es un conjunto (no necesariamente finito) de estados
- T es un conjunto (no necesariamente finito) de transiciones, en nuestro caso de eventos.
- Rel es un conjunto de $Q \times T \times Q$

Nos proponemos construir un sistema de transición de estados para cada descripción de procesos.

Sean $p = < L, I, R >$ (donde $L = < D, L >$ con L conteniendo F,K,E)

$H_F(p) = \{c / var(c) = \emptyset \wedge c \in T_d \in T(L)$ para cada $d \in D\}$ el *universo de Herbrand de las operaciones en* p

La *saturación* en $H_F(p)$ *de una regla* $t_1 \xrightarrow{t_2} t_3 \in R$ es el conjunto

$$Sat_{H_F(p)}\ (t_1 \xrightarrow{t_2} t_3) = \{e_1 \xrightarrow{e_2} e_3 / e_1 = \theta t_1 \wedge e_2 = \theta t_2 \wedge e_3 = \theta t_3 \wedge \theta : var(t_1) \cup var(t_2) \to H_F(p)\}$$

Sean $Universo(R) = \bigcup_{\zeta \in R} Sat_{H_F(p)}(\zeta)$

$Estados(R) = \{e_1, e_3 / e_1 \xrightarrow{e_2} e_3 \in Universo(R)\} \cup I$

$Eventos(R) = \{e_2 / e_1 \xrightarrow{e_2} e_3 \in Universo(R)\}$

La *evolución de la descripción* p es una relación

$$\rightsquigarrow_n \subseteq Estados(R) \times Eventos(R) \times Estados(R)$$

definida por (se denota a $\overset{b}{\rightsquigarrow}_n$ c para $<a,b,c> \in \rightsquigarrow_n$):

i. $e_1 \overset{e_2}{\rightsquigarrow}_0 e_3$ ssi $e_1 \xrightarrow{e_2} e_3 \in Universo(R) \wedge e_1 \in I$

ii. $e_1 \overset{e_2}{\rightsquigarrow_n} e_3$ ssi $e_1 \overset{e_2}{\longrightarrow} e_3 \in \text{Universo}(R) \wedge \exists\, e_1' \overset{e_2'}{\underset{n-1}{\rightsquigarrow}} e_1$ 532

Entonces, el sistema de transición de estados de una descripción de procesos p es el siguiente

$$S(R) = (\text{Estados}(R), \text{Eventos}(R), \rightsquigarrow)$$

donde $\rightsquigarrow = \bigcup_{n\in\mathbb{N}} \rightsquigarrow_n$

2.2.2 LA DESCRIPCION DE UN PROCESO: UN SISTEMA DE TRANSICION DE TERMINOS.

El estudio de ciertas propiedades sobre los procesos es mas facilmente realizable si se posee una interpretación mas "finita" de su comportamiento.
Así por ejemplo las Redes de Petri o los Programas Paralelos [Kel76] son formalismos operacionales para tener una representación finita del comportamiento infinito. Por ello nosotros proponemos un segundo modelo operacional que interpreta a $p = \langle L, I, R \rangle$ como un Sistema de Transición de Términos con Eventos. Se pudiera decir que el primer modelo es el conceptual mientras que el segundo es su interpretación finita.

Sea $H_{F(p)}$ el universo de Herbrand de las operaciones en $p = \langle L, I, R \rangle$

$$\forall t \in T(L) \text{ se define } Sat_{H_F(p)}(t) = \{t' / \theta t = t' \wedge \theta : var(t) \to H_F(p)\}$$

En el caso que $t \in T_{estado}$ entonces $Sat_{H_F(p)}(t)$ contiene todos los estados denotados por el término t.

$$\text{Estados}(p) = \bigcup_{t\in T_{estados}} Sat_{H_F(p)}(t)$$

Una regla $t_1 \overset{t_2}{\longrightarrow} t_3 \in R$ se dice *aplicable* a partir $e \in \text{Estados}(p)$ si existe $\theta : var(t_1) \cup var(t_2) \to H_F(p)$ t.q. $\theta t_1 = e$.

Una vez que esta regla es aplicada el evento que se produce es θt_2 y el estado resultante es θt_3.

Finalmente, si denotamos por $\text{Eventos}(p) = \bigcup_{t\in T_{evento}} Sat_{H_F(p)}(t)$

La relación calculada por la descripción de un proceso $p = \langle L, I, R \rangle$ es

$$\vdash = \bigcup_{n\in\mathbb{N}} \vdash_n \subseteq \text{Estados}(p) \times \text{Eventos}(p) \times \text{Estados}(p)$$

donde $e_1 \overset{e_2}{\vdash_0} e_3$ ssi existe $t_1 \overset{t_2}{\longrightarrow} t_3 \in R$ y $e_1 \in I$ t.q. la regla es aplicada a partir del estado e_1 i.e. existe $\theta : var(t_1) \cup var(t_2) \to H_F(p)$

t.q. $\theta t_1 = e_1$, $\theta t_2 = e_2$, $\theta t_3 = e_3$

$e_1 \overset{e_2}{\vdash_n} e_3$ ssi existe $t_1 \overset{t_2}{\longrightarrow} t_3 \in R$ y $e_1' \overset{e_2'}{\vdash_{n-1}} e_1$ t.q. la regla es aplicada a partir del estado e_1

Se puede probar por inducción sobre n que $\sim\sim^{n} = \Gamma$

3. LA CONSTRUCCION DE UN SPC.

La especificación de sistemas de procesos que se comunican supone la posibilidad de "combinar" otros procesos especificados. Esto permite un grado de abstracción que facilita la "construcción" de sistemas muy complejos.

Para definir esta combinación, C.A.R. Hoare [Hoa 81a], R. Milner [Mil 81] W.C. Rounds-S.D. Brookes [Rou-Bro 81] utilizan las algebras de procesos.
Una expresión en estas algebras:

- tiene como operandos expresiones del algebra.
- describe un sistema de procesos, i.e. el "sentido" de una expresión es en
  [Hoa 81a] un conjunto de historias,
  [Mil 81] un árbol de sincronización o de comunicación,
  [Rou_Bro 81] un conjunto de historia y un árbol

En el modelo que nosotros presentamos aquí, se define tambien un álgebra de expresiones. Esta álgebra esta fundada sobre las ideas de Ph. Jorrand expuestas en [Jor 81, Jor 84]. En esta algebra a diferencia de las anteriores, una expresión

- tiene también como operandos expresiones del algebra
- da como resultado una descripción de procesos

Es decir esta álgebra es simplemente una manipulación simbólica de descripciones. Para conocer el "sentido" de una expresión es necesario interpretar la descripción resultante por una de las semánticas dadas para el lenguaje formal.
De esta forma todo sistema de procesos se reduce a un proceso simple.

3.1. EL ALGEBRA DE DESCRIPCIONES: CALCULO DE JORRAND.

Puesto que el álgebra que se va a definir es una manipulación formal de descripciones, nosotros vamos a utilizar tres funciones Ψ,Φ,Π que nos permitan hacer ciertas combinaciones de símbolos y definir el "producto de estados" y el "producto de términos de estados" necesarios para la descripción de los operadores del álgebra.

Sea Ψ una combinación arbitraria de secuencias finitas (por ejemplo la concatenación de secuencias).

Sea E_1,E_2 dos conjuntos de símbolos de estados con funciones de aridad ρ_{E_1},ρ_{E_2}
El *producto de estados* $E_1 \times E_2$ es un conjunto de símbolos de estados $\hat{E}$ con función de aridad $\rho_{\hat{E}}$ t.q. existe una función $\phi : E_1 \times E_2 \leftrightarrow \hat{E}$ donde

si $\phi(e_1,e_2) = e$ entonces $\rho_{\hat{E}}(e) = \Psi(\rho_{E_1}(e_1),\rho_{E_2}(e_2))$

El *producto de términos de estados* $T_{estado_{E_1}} \times T_{estado_{E_2}}$ es un conjunto de términos de estados definidos por la función

$$\Pi_{E_1 x E_2} : T_{estado_{E_1}} \times T_{estado_{E_2}} \rightarrow T_{estado_{E_1 x E_2}}$$

t.q.

$$\Pi_{E_1 x E_2}\ (e(t_1,\ldots,t_n)\ ,\ e'(t'_1,\ldots,t'_m)) = e_1(\psi(<t_1,\ldots,t_n>,<t'_1,\ldots,t'_m>))$$

donde $\phi(e,e') = e_1$

El algebra de descripciones puede ahora ser definida como la estructura algebraica

$$< A\ ,\ \|\ ,\ +\ ,\ - >$$

donde

- A es el conjunto de todas las descripciones de procesos $p = <L,I,R>$
- los operadores son definidos por:

El operador unión $= \| : A \times A \rightarrow A$

Este operador tiene por objetivo poner en paralelo sus procesos operados denotados por las descripciones de procesos p_1 y p_2. Así la descripción del proceso resultante de evaluar la expresión $p_1 \| p_2$ puede comportarse en cada etapa de su evolución:

1. Sea, como el proceso p_1 evolucionando solo i.e. dejando el proceso p_2 en el mismo estado.
2. Sea, como el proceso p_2 evolucionando solo i.e. dejando el proceso p_1 en el mismo estado.
3. Sea, como los dos procesos p_1 y p_2 evolucionando simultaneamente

Formalmente

Sea $p_1 = < L_{p_1}, I_{p_1}, R_{p_1}>$ donde $L_{p_1} = < D_{p_1}, L_{p_1}>$ con L_{p_1} compuesto de $F_{p_1}, K_{p_1}, E_{p_1}$

Sea $p_2 = < L_{p_2}, I_{p_2}, R_{p_2}>$ donde $L_{p_2} = < D_{p_2}, L_{p_2}>$ con L_{p_2} compuesto de $F_{p_2}, K_{p_2}, E_{p_2}$.

Los tres conjuntos de reglas siguientes describen los tres puntos mencionados anteriormente

$$\hat{R}_1 = \{\Pi(t_1,t'_1) \overset{t}{\dashv^2} \Pi(t_3,t'_1)/t_1 \overset{t}{\dashv^2} t_3 \in R_{p_1} \wedge (t'_1 = S(x_1,\ldots,x_n) \wedge \rho_{E_{p_2}}(s) = d_1 \ldots d_n \wedge x_1,\ldots,x_n \notin var(t_1 \overset{t}{\dashv^2} t_3)) \text{ para cada } s \in E_{p_2}\}$$

$$\hat{R}_2 = \{\Pi(t_1,t'_1) \overset{t'}{\dashv^2} \Pi(t_1,t'_3)/t'_1 \overset{t'}{\dashv^2} t'_3 \in R_{p_2} \wedge (t_1 = s(x_1,\ldots,x_n) \wedge \rho_{E_{p_1}}(s) = d_1 \ldots d_n \wedge x_1,\ldots,x_n \notin var(t'_1 \overset{t'}{\dashv^2} t'_3)) \text{ para cada } s \in E_{p_1}\}$$

$$\hat{R}_3 = \{\Pi(t_1,t_1') \xrightarrow{t_2 \cup t_2'} \Pi(t_3,t_3')/t_1 \xrightarrow{t_2} t_3 \in R_{p_1} \wedge t_1' \xrightarrow{t_2'} t_3' \in R_{p_2}\}$$

Luego la descripción definida por $p_1 \parallel p_2$ es la siguiente:

$p_1 \parallel p_2 = <\hat{L},\hat{I},\hat{R}>$ donde $\hat{L} = <D_{p_1} \cup D_{p_2}, \hat{L}>$ con $\hat{L}$ compuesto de

$F_{p_1} \cup F_{p_2}$, $K_{p_1} \cup K_{p_2}$, $E_1 \times E_2$

$\hat{I} = \{\Pi(e_1,e_2)/e_1 \in I_{p_1} \wedge e_2 \in I_{p_2}\}$

$\hat{R} = \hat{R}_1 \cup \hat{R}_2 \cup \hat{R}_3$

El operador de conexión $+ : A \times K_{IN} \to A$

Sea $p = <L,I,R>$ donde $L = <D,L>$ con L compuesto de F,K,E.

El operador de conexión construye un nuevo proceso en el cual dos puertas (definidas sobre el mismo dominio), una de entrada $\delta.i$ y otra de salida $o.\delta$ son conectadas. El nuevo proceso permite así todas las comunicaciones del antiguo proceso, mas las comunicaciones sobre $o.i$ en los estados en los cuales $\delta.i$ y $o.\delta$ son posibles simultaneamente.

Formalmente

$$p + o.i = <\hat{L},\hat{I},\hat{R}>$$

donde

$\hat{L} = <D,\hat{L}>$ con $\hat{L}$ conteniendo $F,\hat{K} = K \cup \{o.i\}$, E

donde $\rho_{\hat{K}}(k) = \rho_K(k)$ para $k \in K$

y $\rho_{\hat{K}}(o.i) = \rho_K(\delta.i) = \rho_K(o.\delta)$

$\hat{I} = I$

$\hat{R} = R \cup \{\theta t_1 \xrightarrow{\theta t_2'} \theta t_3/t_1 \xrightarrow{t_2} t_3 \in R \wedge$

$\exists\, t\, u \in T_{\rho_{\hat{K}}(o.i)}$. $o.\delta(t) \in t_2 \wedge \delta.i(u) \in t_2$

$\wedge\ t_2' = t_2 - \{o.\delta(t), \delta.i(u)\} \cup \{i.o(\theta t)\}$

$\wedge\ \theta = pgu(t,u)\}$

$pgu(t,u)$ denota el mas grande unificador de t y u, i.e. una substitución $\theta : var(t) \cup var(u) \to T_{ope}$ t.q. $\theta t = \theta u$ es llamado un unificador de t y u. Si se tienen dos unificadores θ' y θ'', se dice que θ'' es mas general que θ', si existe una substitución θ t.q. $\theta' = \theta \circ \theta''$. El pgu (t,u) es calculado por el algoritmo de unificación [Rob 65,71].

El operador de restricción $- : A \times K \to A$

Sea $p = <L,I,R>$ donde $L = <D,L>$ con L compuesto por F,K,E.

El objetivo de este operador de restricción es construir un nuevo proceso en el cual la conexión k, (argumento del operador) se vuelve invisible. Esto quiere decir, en

el caso que:

1. $k \in K_{EX}$, el nuevo proceso no permite ninguna comunicación por esa puerta.
2. $k \in K_{IN}$ el nuevo proceso puede efectuar las comunicaciones a través de la conexión k , pero ellas no pueden ser observadas por el medio exterior.

Formalmente

$p\text{-}k = < \hat{L}, \hat{I}, \hat{R} >$ donde

$\hat{L} = < D, \hat{L} >$ con $\hat{L}$ conteniendo F , $\hat{K} = K - \{k\}$, E

$\hat{I} = I$

$\hat{R} = (R - R') \cup R''$

con

$$R' = \{t_1 \xrightarrow{t_2} t_3 / t_1 \xrightarrow{t_2} t_3 \in R \wedge \exists\, u \in T_{\rho_{\hat{k}(k)}} \cdot k(u) \in t_2\}$$

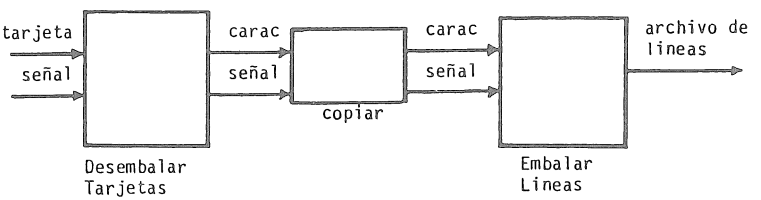
$$R'' = \{t_1 \xrightarrow{t_2 - \{k(u)\}} t_3 / t_1 \xrightarrow{t_2} t_3 \in R \wedge k \in k_{IN} \wedge k(u) \in t_2\}$$

4. LA POTENCIA DEL LENGUAJE Y EL ANALISIS DE SPC.

El algebra de descripciones $< A, \| , +, - >$ tiene un interés multiple, asÍ:

1. el algebra reduce el estudio de un SPC al estudio de un proceso simple.
2. el diseño de un SPC es simple y claro : se describen los procesos elementales y utilizando los operadores, se obtienen sistemas complejos, sin necesidad de dar su descripción.

   Por ejemplo la expresión asociado a



describe el proceso de pasar archivos de tarjetas de 80 caracteres a archivos de lineas de 125 caracteres cada una. Cada tarjeta esta seguida de un espacio en blanco y la Gl_ tima linea esta completada por espacios blancos si es necesario.

3. en [Jor 84] se muestra que con las bases de esta álgebra, se pueden descri_ bir los operadores del calculo de R. Milner [Mil 80] , C.A.R. Hoare - S. D. Brookes - A. W. Roscoe [Hoa-Bro-Ros 81] , Austry-Boudol [Aus-Bou 82] y W. C. Rounds - S. D. Brookes [Rou-Bro 81] .
4. en [Per 84] se demuestra que con este lenguaje se pueden expresar todas las funciones calculables.

5. en [Jor 84a][Per 84] se interpreta la sintaxis formal de los procesos $p = < L, I, R >$ como la presentación de una variedad de algebras. Esta semántica permite analizar los procesos en el mismo sentido de los trabajos realizados en tipos abstractos de datos. Esta semántica algebraica da las bases necesarias para introducir este lenguaje en una extensión del lenguaje de tipos abstractos LPG (Langage de Programmation Générique) [Ber 79, 82a, 82b].

6. La semántica operacional dada al lenguaje pensamos que permite el estudio de propiedades tales como: invariantes, trayectorias, terminación, "deadlock",... en el estilo de las investigaciones realizadas por J. Sifakis [Sif 80], sobre los sistemas de transición de estado.

7. La propiedad de divergencia i.e. la posibilidad, para el proceso de estar calculando indefinidamente sin comunicarse con el medio exterior, es indecidible. Una demostración de esto se encuentra en [Per 84]. Sin embargo nosotros creemos que existe una gran familia para la cual esta propiedad es decidible. Una via de como decidir esta propiedad se encuentra en [Per 84] a traves del estudio de funciones noetherianas y su relación con el problema de decidabilidad de terminación de una regla de transición de términos.

BIBLIOGRAFIA

[Aus-Bou 82] D. Austry - G. Boudol
"Algèbre de Processus et Synchronisation"
A ser publicado en TCS

[Ber 79] D. Bert
"La programmation Générique"
Thèse d'Etat (Juin 1979). Université de Grenoble.

[Ber 82a] D. Bert
"Refinements of Generic Specifications with Algebraic Tools"
R. R. 335, IMAG 1982.

[Ber 82b] D. Bert
"Generic Programming: A tool for Designing Universal Operator Application to Program Algebra"
R. R. 336, IMAG 1982.

[Hoa 78] C.A.R. Hoare
"Communication Sequential Processes"
C.A.M. (Vol. 21, Nº 8), 1978.

[Hoa-Bro-Ros 81] C.A.R. Hoare - S.D. Brookes - A. W. Roscoe
"A theory of Communicating Sequential Processes"
PRG Monograph N°16, 1981 - Oxford University.

[Hoa 81a] C.A.R. Hoare
"A model for Communicating Sequential Processes"
PRG Monograph N°22, 1981 - Oxford University.

[Jor 81a] Ph. Jorrand
"Bases for the Specification of Communicating Processes"
ICS 81, London - April 1981.

[Jor 81b] Ph. Jorrand
"Specification of Communicating Processes and Process Implementation Correctness"
International Symposium on Programming.
5th Colloquium, Turin - April 1982 Proceedings LNC 137.

[Jor 84a] Ph. Jorrand
"Communicating Processes as Term Algebras"
A ser publicado

[Jor 84b] Ph. Jorrand
"FP2: Functional Parallel Programming Based on Term Substitution"
A ser publicado.

[Mil 80] R. Milner
"A calculus of Communicating Systems"
LNCS 92, 1980.

[Per 84] J. M. Pereira
"Processus Communicants: un langage formel et ses modèles. Problemes d'Analyse".
Thèse de 3 ème cycle (Juin 1984).
Université de Grenoble

[Rob 65] J. A. Robinson
"A Machine-Oriented Logic Based on the Resolution Principle"
JACM 12, 1965.

[Rou-Bro 81] W.C. Rounds - S. D. Brookes
"Possibles Futures, Acceptances, Refusals and Communicating Processes".
Proceedings of 22nd Annual IEEE Symposium on Foundations of Computer Science, 1981.

[Sif 82] J. Sifakis
"An Unified Approach for Studying the Properties of Transition Systems"
TCS 18 (1982)

[Kel 76] R.M. Keller
"Formal verification of Parallel Programs"
CACM (Vol. 19 N°7) 1976

SUPORTE DE EXECUÇÃO PARA A LINGUAGEM LANAC-II, UTILIZANDO AS FACILIDADES DO DOS-8000 DO SISCO MB-8000

V.A.D. Oleinki*
O.B. Coelho**

SUMÁRIO

Este trabalho descreve o projeto e a implementação do suporte de execução da linguagem LANAC-II. Este suporte de execução está sendo elaborado utilizando as facilidades do DOS-8000, que é um sistema operacional orientado para aplicações em tempo real, disponível no equipamento SISCO MB-8000.

ABSTRACT

This work describes the project and implementation of LANAC-II execution support. This execution support is being elaborated using the DOS-8000 facilities, an operating system for real time use, available in the SISCO MB-8000 computer.

* Bacharel em Ciência da Computação (UFSCar, 1979). MSc. em Engenharia de Sistemas e Computação (COPPE/UFRJ, 1984). Áreas de interesse: compiladores, linguagens de programação, sistemas operacionais. Atualmente Assistente de Pesquisa na Divisão de Pesquisa e Desenvolvimento do Departamento de Informática - Instituto de Pesquisas Espaciais (INPE) - Av. dos Astronautas, 1758 - Caixa Postal 515 - 12200 - São José dos Campos - SP.

** Bacharel em Matemática (UFRJ, 1977). MSc. em Matemática Pura (UFRJ, 1982). Áreas de interesse: redes de computadores, sistemas operacionais. Atualmente Assistente de Pesquisa na Divisão de Pesquisa e Desenvolvimento do Departamento de Informática - Instituto de Pesquisas Espaciais (INPE) - Av. dos Astronautas, 1758 - Caixa Postal 515 - 12200 - São José dos Campos - SP

A linguagem LANAC-II (Linguagem de Alto Nível para Aquisição de Dados e Controle) [1] é voltada para sistemas de tempo real e processamento concorrente. Ela foi desenvolvida no INPE/CNPq como uma ferramenta para as atividades de aquisição, processamento e disseminação de dados enviados por satélites meteorológicos.

Nas implementações anteriores desta linguagem para os equipamentos HP2116 B e PDP 11/10, o suporte de execução, denominado "Cerne", constituia-se em um conjunto de rotinas e estruturas de dados que tinham por objetivo substituir os sistemas operacionais das referidas máquinas, possibilitando a execução dos comandos da linguagem.

A versão atual da linguagem está sendo implementada para o equipamento SISCO MB-8000, que possui o sistema operacional DOS-8000 [2], orientado para aplicação em tempo real; assim, o atual suporte de execução não será mais "stand alone", o que permitirá que se utilize todo o potencial de "software" disponível no referido sistema.

O sistema operacional DOS-8000 tem a capacidade de ordenar e alocar o controle para diversas tarefas que competem ao processador. O RDOS, escalonador de tarefas, mantém informações sobre o estado de cada tarefa em uma estrutura denominada TCB (TASK CONTROL BLOCK). A operação de supervisão das tarefas consiste essencialmente nas modificações das filas de TCBs e da alocação de recursos à tarefa de maior prioridade que esteja apta a prosseguir com o seu processamento.

As tarefas podem assumir quatro estados: "prontas" - que estão aptas a serem executadas; "em execução" - que estão de posse do processador; "suspensas" - que estão inabilitadas temporariamente para receber o processador; "dormentes" - que não estão aptas a receber o processador até que sejam colocadas no estado "prontas", através de um comando específico.

O mecanismo de sincronização e comunicação entre tarefas, disponível no DOS-8000, é através de troca de mensagem.

De modo a permitir toda a manipulação de tarefas em um ambiente multitarefas, o sistema operacional DOS-8000 oferece diversas primitivas; pode-se ressaltar as seguintes: "INICIA TAREFA (.TASK)" - coloca a tarefa no estado "prontas", podendo especificar uma prioridade; "TERMINA TAREFA (.KILL)" - coloca a tarefa no estado "dormentes"; "SUSPENDE TAREFA (.SUSP)" - coloca a tarefa no estado "suspensas"; "TRANSMITE MENSAGEM (.XMT)" - envia uma palavra de mensagem a outra tarefa; "ESPERA MENSAGEM (.REC)" - aguarda o recebimento de uma palavra de mensagem.

Através dessas facilidades fornecidas pelo DOS-8000, o suporte de execução da linguagem LANAC-II está sendo implementado através de estruturas de dados e rotinas que utilizam chamadas a essas primitivas, de modo a formar uma extensão a esse sistema operacional, permitindo a execução dos comandos da linguagem.

Os mecanismos de sincronização e comunicação entre as tarefas da linguagem LANAC-II são: semáforos, regiões críticas simples e condicionais. Uma das característi

cas relevantes no mecanismo de sincronização pretendido é a hierarquia entre as tarefas, com o seguinte significado: quando uma tarefa (dita "PAI") inicializa outras (ditas "FILHOS"), ela deve ser suspensa até que as inicializadas terminem.

Para satisfazer esse comportamento e essas estruturas, foi criado um descritor de processos de modo a implementar a hierarquia, que o sistema DOS-8000 não considera, e as filas de semáforos e regiões críticas; cada tarefa possui uma entrada neste descritor, cujos campos contêm as seguintes informações: - identidade da tarefa "PAI" - indica qual a tarefa que inicializou a tarefa em questão; - número de "FILHOS" da tarefa - indica o número de tarefas que esta inicializa; - Ponteiro - utilizado para encadear as tarefas que estão na fila à espera de semáforos e regiões críticas.

Desse modo, a primitiva "INICIE PROCESSO" da linguagem LANAC-II é implementada do seguinte modo:

```
INÍCIO

Se nrodefilhos (identidadedopai) = 0
então elevaprioridade (identidadedopai);
nrodefilhos (identidadedopai) = nrodefilhos (identidadedopai) + 1;
identidadedatarefapai (identidade do filho) = identidadedopai;
iniciatarefa (identidadedofilho);

FIM
```

O suporte de execução é responsável pelas operações de E/S, devendo acionar os "drivers" dos periféricos e atender às suas interrupções. Cada periférico é representado por um descritor, com a finalidade de implementar os comandos de controle disponíveis na linguagem: (CONTROLES, ESTADOES, EVENTEX, LEIA, REGISTRE). O DOS-8000 não possui muitas facilidades para controle de periféricos, sendo que esta parte do suporte de execução exigirá estudos futuros de modo a permitir uma eficiente implementação destes requisitos da linguagem.

O compilador LANAC-II gera código para o SISCO MB-8000; atualmente é um compilador cruzado, escrito em ALGOL do sistema BURROUGHS 6800, mas já está sendo transladado para a máquina alvo. Este trabalho de translado está sendo elaborado em paralelo com a definição e implementação do suporte de execução.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] FISCHER, S.D. Projeto da Linguagem LANAC-II e sua Implementação para o SISCO MB-8000. Dissertação de Mestrado em Computação Aplicada, SJCampos, INPE, 1984. (INPE-3090-TDL/163).

[2] SISCO, SISTEMAS E COMPUTADORES S.A. DOS-8000 - Manual de Referência, São Paulo, 1980.

M.B. CORREIA*

R.A. FARIA**

A.R.S. SERRA JR.***

E.N. SILVA***

SUMÁRIO

A finalidade deste trabalho é implementar a aritmetica de intervalo no DEC-10 usando os algoritmos desenvolvidos por Kulisch/Miranker, que permitem obter resultados com exatidão máxima, de grande importância, normente em se tratando da resolução de métodos numéricos.

ABSTRACT

The purpose of this work is to implement the interval arithmetics in the DEC-10 utilizing the algorithms developed by Kulisch-Miranker which permit obtaining results with maximum accuracy, what is rather important mainly when dealing with the resolution of numerical methods.

* Prof. Adjunto UFPE (Docteur Ingenieur-INP-Toulouse-France-1983); arquitetura de computadores, eletrônica digital.

** Prof. Assistente UFPE (Mestre em Matemática-UFPE-1971); Matemática Computacional, Análise Numérica, Aritmética de computador.

*** Engenheiro Eletricista (UFPE-84) (Aluno do Mestrado em Informática da UFPE); Eletrônica digital, arquitetura de computadores, aritmética de computadores.

IMPLEMENTAÇÃO DA ARITMÉTICA DE INTERVALO COM EXATIDÃO MÁXIMA

Em computação, um dos problemas fundamentais de análise numérica é a determinação do erro decorrente da utilização da aritmética de ponto flutuante.

Este problema pode ser resolvido exprimindo os resultados através de limites da forma $(x - \xi)$ e $(x + \xi)$ tais que o intervalo por eles definido contenha a solução exata. Esta idéia levou à definição de um novo tipo de número $X = [\underline{X}, \overline{X}]$ chamado intervalo, que representa o conjunto de todos os reais contidos no intervalo definido pelos limites inferior $\underline{X}$ e superior $\overline{X}$.

A aritmética de intervalo é definida de modo natural por:

$$[\underline{X}, \overline{X}] \circ [\underline{Y}, \overline{Y}] = [\underline{Z}, \overline{Z}] \text{ , onde } \circ \in \{+, -, x, /\}$$

com $\underline{Z} = \min_{\substack{x\in[\underline{X},\overline{X}] \\ y\in[\underline{Y},\overline{Y}]}} (x \circ y)$ e $\overline{Z} = \max_{\substack{x\in[\underline{X},\overline{X}] \\ y\in[\underline{Y},\overline{Y}]}} (x \circ y)$.

A computação de $[\underline{Z}, \overline{Z}]$ envolve operações de ponto flutuante que em geral são inexatas; nestes casos, o limite inferior deve ser arredondado por falta e o limite superior por excesso, de modo a garantir que o intervalo resultante contenha a solução exata. Essa política, chamada de arredondamento direcionado, inclui também o tratamento do "overflow" e do "underflow".

Os computadores, de uma maneira geral (em particular, o DEC-10) não prevêem o uso de arredondamento direcionado. Para implementá-lo costuma-se então usar um fator de conversão, igual a $1-1/2\ b^{1-n}$ para máquina que arredonda para o número mais próximo e $1-b^{1-n}$ para máquina que trunca (n é o número de dígitos da mantissa). Este procedimento pode levar a resultados pessimistas, isto é, a intervalos contendo a solução, mas sem muita exatidão.

Os algoritmos implementados nesse trabalho foram desenvolvidos pelo / Kulisch/Miranker e garantem a exatidão máxima. Como sua implementação requer o uso de uma política de arredondamento adequada, além da manipulação de $+\infty$ e $-\infty$, julgou-se conveniente usar a aritmética inteira do DEC-10. Os números de ponto flutuante são então tratados como ponto fixo e padrões especiais são definidos para representar o 0, $+\infty$ e $-\infty$. A ocorrência de "overflow" e "underflow" é assinalada ao programador, ao mesmo tempo que valores prefixados são atribuidos às variáveis, de modo que a computação possa prosseguir.

Este trabalho faz parte de uma pesquisa com objetivo de definir um microcomputador que tenha recursos de hardware e software para tratar a aritmética de intervalo. Nesta fase foi simulado no DEC-10, as operações da aritmética de intervalo e desenvolvidas algumas aplicações que permitem testar o seu desempenho. Em paralelo, foi feito um estudo sobre a viabilidade do uso do sistema INTEL 8086/8087 para a implementação de um microcomputador dedicado à aritmética de intervalo.

BIBLIOGRAFIA

KULISCH, U.W., MIRANKER, W.L., "Computer Arithmetic in Theory and practice", Academic Press, 1981.

MOORE, R.E., "Interval Analysis", Prentice Hall, 1979.

STERBENZ, P.H., "Floating - Point Computation", Prentice Hall, 1974.

CLAUDIO, D.M., "Noções de Matemática Intervalar", 1984.

SUPORTE ELETRÔNICO PARA UM ESPECTROFOTÔMETRO INFRAVERMELHO POR TRANSFORMADA DE FOURIER RÁPIDA ($)

L.F. COSTA*, M.SIU LI**

SUMÁRIO

Este trabalho pretende comunicar sobre a parte eletrônica-computacional que está sendo desenvolvida para um espectrofotômetro por Transformada de Fourier Rápida no Instituto de Física e Química de São Carlos-USP. Apresentamos uma descrição suscinta do princípio de funcionamento, seguida da descrição dos módulos computacionais.

ABSTRACT

This paper is intended to be a communication about the electronic-computational part of a Fast Fourier Transform Spectrophotometer in development at Instituto de Física e Química de São Carlos-USP. A brief physical description preceeds the exploration of the computational modules.

* Engenheiro de Eletrônica (EESC-USP, 1984); Projeto e implementação de Sistemas Digitais, Microcomputadores e Software; Instituto de Física e Química de São Carlos-USP - Av. Dr. Carlos Botelho 1465, São Carlos, SP, CEP 13.560.

** Professor Assistente Doutor (MS-3); Coordenador do Projeto "Construção de protótipo de espectrofotômetro para infravermelho; Instituto de Física e Química de São Carlos-USP - Av. Dr. Carlos Botelho 1465, São Carlos, SP, CEP 13.560.

$ Projeto financiado pela FIPEC - número 1291

A espectroscopia por Transformada de Fourier apresenta diversas vantagens sobre as técnicas tradicionais (dispersivas) para a obtenção de espectros de transmissividade x comprimento de onda de amostras. Entretanto, apenas com o avanço da microeletrônica foi possível o desenvolvimento e utilização em larga escala de instrumentos deste tipo. O presente instrumento pretende operar no intervalo espectral de 4000 - 400 cm^{-1} (2,5 a 25 microns) com uma resolução estimada de 4 a 5 cm^{-1}.

A partir de um sistema ótico-eletro-mecânico (um interferômetro de varredura) obtém-se um padrão de interferência entre dois feixes de infravermelho que atravessam uma amostra do material a ser analisado. Este sinal, o INTERFEROGRAMA, é então discretizado e a Transformada de Fourier (método numérico FFT - "Fast Fourier Transform") destes valores resulta num espectro. O processo é repetido sem a amostra e o espectro das transmissividade da amostra é obtido como a razão entre os dois espectros.

Nosso sistema eletrônico constitui-se de três módulos básicos interconectados através de um controlador de duto:

1- Microcomputador Central de Controle
2- Sistema de Aquisição de Dados microcontrolado
3- Processador dedicado para aplicação do FFT.

Incluímos também um Terminal de Vídeo, Impressora, Registrador CRT, um Programador de "EPROMs"/Emulador para apoio e todo o "software" numérico e de controle.

O módulo 1, encarregado do controle do sistema, é um micro computador convencional com CP/M e microprocessador Z-80A mais interface serial (8251, 8253), controlador de "Disk-drivers" (2793), Controle do Duto, Interface Registrador X-Y CRT e "light-pen". Têm-se ainda 4KBytes de monitor (2732) e 64KBytes de Memória Dinâmica.

O módulo 2, também com Z-80A, constitui-se de Interface Paralela (8255) para o Sistema Ótico, Conversor D/A 12 Bits (AD-574) com "Track-Hold", contém as rotinas de controle do Sistema Ótico em assembler para amostragem sincronizada e pré-processamento que resultarão no Interferograma armazenado num "buffer" de 8KBytes locais de memória estática RAM.

O módulo 3, após receber o Interferograma via Micro de Controle, aplica o FFT, resultando no Espectro que retornará ao módulo 1, onde providencia-se o "display" no Registrador CRT e/ou na Impressora Gráfica e permite-se uma série de consultas do operador, via teclado ("Zoom", valores de pontos selecionados, etc.)

O sistema permanece em processamento (todos os módulos) até que os resultados, obtidos por média ("average") dos espectros, sejam considerados satisfatórios.

Nosso instrumento encontra-se em desenvolvimento, estando os módulos 1 e 2 completamente definidos e em fase final de testes (aplicação do FFT em FORTRAN no módulo de Controle), enquanto o módulo 3 ainda é objeto de estudos.

Futuramente, pretendemos incluir outras opções como um videográfico, sendo que a natureza modular do sistema possibilitará ao usuário liberdade para a escolha de sua configuração.

M.G.V. NUNES*

SUMARIO

O Método de Harrison [2] para a busca exata de subcadeias de caracteres em textos utiliza o conceito de assinaturas associadas ao texto e às subcadeias procuradas. Com isso, além de compactar sensivelmente o texto original, sob esse processo nem todo o texto precisa ser exaustivamente pesquisado, isto é, apenas aqueles trechos do texto original onde há uma alta probabilidade de ocorrência da subcadeia são realmente pesquisados caracter por caracter.
Este trabalho apresenta dois tipos de funções-assinatura amplamente testadas e os resultados experimentais obtidos.

ABSTRACT

Harrison's Method [2] uses text signatures in String Searching. With this technique there is a sensible compactation of the original text and is not necessary to search the text wxhaustively. The text signature is examined first to determine if there is a high probability of finding the sought pattern and only in this case the correspondent line of text is scanned.
In this work we study and test in deep two types of text signatures and show the experimental results obtained.

*Bacharel em Ciências da Computação (UFSCar-1980). Mestre em Ciências da Computação (USP-1985); algoritmos, sistemas especialistas; ICMSC-USP - Caixa Postal, 668 - São Carlos, SP - CEP 13560; Tel:(0162) 712214.

INTRODUÇÃO

O problema de detetar a (não) ocorrência de uma ou várias cadeias de caracteres num texto caracteriza uma parte importante de muitos problemas relacionados a processamento de textos como edição de textos, recuperação de informações, manipulação de símbolos e pesquisa bibliográfica.

Vamos analisar, neste trabalho, um método de busca apropriado para textos grandes e pouco dinâmicos, uma vez que é realizado um pré-processamento do texto, codificando-o e compactando-o, e qualquer alteração no texto original requereria um novo pré-processamento. O objetivo principal do método é pesquisar exaustivamente uma parte do texto (por exemplo, uma linha) apenas quando houver uma grande probabilidade de se encontrar a cadeia neste trecho. Neste caso, qualquer método direto de busca pode ser utilizado (ressaltamos o método de Boyer e Moore [1]).

O MÉTODO DE HARRISON

A informação contida em cada linha do texto é condensada num código de w bits que passa a ser a assinatura da linha. O código é obtido aplicando-se uma função hash h a todos os grupos consecutivos de k símbolos, produzindo um inteiro entre 0 e w-1. O bit correspondente ao inteiro produzido torna-se 1. Esta cadeia binária resultante, que contém 1 nas posições que correspondem a certas subcadeias de comprimento k, é chamada k-assinatura da linha do texto. O mesmo procedimento é efetuado para obter a k-assinatura da cadeia a ser procurada. Compara-se então, a assinatura da cadeia com as assinaturas das linhas e apenas quando este teste sugerir uma provável correspondência na linha associada, é que esta linha original será realmente pesquisada.

ESCOLHA DOS PARÂMETROS w E k E DA FUNÇÃO HASH h

Para um texto sobre um alfabeto de 64 símbolos, tomamos w, o tamanho da assinatura, múltiplo de 8 $_$ 64 e 128. Em nossos experimentos, tomamos k = 2 e 3 para a busca de cadeias com mais de 1 e 2 caracteres, respectivamente. Lembramos que $k > 1$ nos fornece informações sobre a posição do caracter na linha.

A função sugerida por Tharp e Tai [4] exige que os símbolos do texto sejam divididos em classes baseadas na frequência desses símbolos no texto, tal que as probabilidades de ocorrência são aproximadamente iguais para cada classe. Para um par de símbolos y_1y_2, a função hash é $h(y_1y_2)$ = número-de-classes * $T[y_1]$ + $T[y_2]$,

onde o número-de-classes é escolhido como sendo o maior inteiro, m, tal que $m^k \leq w$ e $T[y_1]$ fornece o número da classe ($0 _ m$-1) a qual pertence o símbolo y_1.

Observamos que quanto maior o alfabeto maior deve ser w, para obtermos um maior número de classes de símbolos.

A segunda função hash testada é uma função simples que, ao contrário da anterior, não requer qualquer informação prévia sobre o texto. Ela simplesmente toma a somatória das or-

dens dos caracteres da subcadeia de comprimento k no comjunto de caracteres adotado pelo compilador Pascal, módulo o comprimento da assinatura.

RESULTADOS EXPERIMENTAIS

Para grupos de 50 cadeias arbitrárias de comprimentos variando de 2 a 15 caracteres, obtivemos o número médio de linhas retidas no teste de assinaturas, e portanto pesquisadas exaustivamente, e dentre estas, o número médio de linhas em que a busca resultou em sucesso, ou seja, onde a cadeia foi encontrada.

De uma maneira geral, 3 fatores influenciaram no comportamento do método: o comprimento da assinatura, w, a função hash aplicada e o comprimento da cadeia procurada.

Quanto maior a assinatura, menor o número de linhas pesquisadas exaustivamente, no sentido de que menor número de cadeias são identificadas incorretamente como subcadeias pelo teste de assinaturas. Em consequência disso, temos um correspondente acréscimo no número de linhas com sucesso. Quanto maiores as cadeias procuradas, mais raramente elas ocorrem no texto e portanto, um menor número de linhas é retido no teste de assinaturas.

A função hash mais simples, que não exige informações sobre o texto, mostrou-se em média inferior à de Tharp e Tai quanto maior a assinatura adotada, pois neste caso, o número de classes de símbolos também é maior. Mas, no caso $k = 3$, para assinaturas menores (64 bits) a função mais simples comportou-se em média, de maneira superior. Isto se deve ao fato de que no caso $k = 3$ o número de classes de símbolos para a função de Tharp e Tai diminuiu causando com isso um acréscimo na probabilidade de ocorrência de cada classe. Assim, quando dispusermos de espaço para assinaturas longas e de informações sobre frequências de símbolos no texto, optamos pela função de Tharp e Tai. Se, por outro lado, não tivermos informações sobre o texto ou necessitarmos usar assinaturas menores ou ainda se desejarmos procurar por cadeias com mais de 2 caracteres, devemos optar por uma função mais simples, como a descrita anteriormente.

Mais detalhes sobre o experimento bem como os gráficos que ilustram os resultados obtidos, encontram-se em [3-Cap.5].

BIBLIOGRAFIA

[1] Boyer,R.S., Moore,J.S. - "A Fast String Searching Algorithm" - Comm. of ACM - Vol. 20 - nº 10 - pp. 762-772 - Out.77.

[2] Harrison,C.M. - "Implementation of the Substring Test by Hashing" - Comm. of ACM - vol. 14 - nº 12 - pp.777-779 - Dez.71.

[3] Nunes,M.G.V. - "Algoritmos de Busca em Texto" - Dissertação de Mestrado - ICMSC-USP, São Carlos, SP - Jan.85.

[4] Tharp,A.L., Tai,K. - "The Practicality of Text Signatures for Accelerating String Searching" - Software-Practice and Experience - vol. 12 - pp. 35-44 - 1982.

ISIS : UN SISTEMA DE INFORMACION ESPACIAL INTELIGENTE.

J. BOSAANS *

RESUMEN

Se describe la filosofía de diseño de un sistema de información espacial inteligente cuyo objetivo es determinar los sitios más apropiados para explotar yacimientos de lignito en el estado de Louisiana, EE.UU., basado la factibilidad económica y ambiental, de una forma altamente interactiva, transparente y "amigable al usuario".

ABSTRACT

The design filosophy of an intelligent spatial information system is described. Its goal is to determine the best suited sites for lignite development in the state of Louisiana, U.S.A., based on the economic and environmental feasibility in a highly interactive, transparent and "user-friendly" fashion.

* Ingeniero Naval Electricista (APN, 1974), Analista de Sistemas (PLANACAP, 1979), - Master of Science in Systems Science (LSU, 1983). Ingeniería de Software, Inteligencia Artificial, Computación gráfica. Armada de Chile. Casilla 453 Fono 978212 Valparaíso, CHILE.

INTRODUCCION

Una de las áreas más interesantes de la investigación en Inteligencia Artificial es la de los llamados Sistemas Expertos o Sistemas basados en el conocimiento.

Estos sistemas no son otra cosa que programas computacionales que por preveer soluciones a importantes problemas a nivel de expertos son heuristicos, transparentes y flexibles.

ISIS selecciona los sitios más adecuados para desarrollar la extracción de lignito en el estado de Louisiana U.S.A. de acuerdo a su factibilidad económica, del medio ambiente y tamaño tal que garantice su conveniencia.

DEFINICION DEL PROBLEMA

El Estado de Louisiana, U.S.A., tiene varios depósitos de lignito conocidos que se piensa son factible de explotar. Estas explotaciones debieran ocurrir en las áreas más adecuadas, es decir aquellos con alta calidad de mineral y que puedan ser explotados a un mínimo costo considerando el impacto social, económico y del medio ambiente.

Se requiere un Sistema de Información que,premunido del "conocimiento"de un experto en Geología Económica, sea capaz de analizar los diversos mapas del Estado de Louisiana y procesar la información relevante que de ellos extraiga, con el propósito de determinar las áreas más conveniente de desarrollar, de acuerdo a criterios técnico, económico y social. El sistema debe, finalmente, determinar la factibilidad comparativa de desarrollar los depósitos de mineral existentes.

SOLUCION POR MEDIO DE ISIS

Una de las principales preocupaciones en el diseño de ISIS, fue la organización de un gran cuerpo de conocimiento que se requiere para hacer la determinación comparativa. Es esencial que el conocimiento geológico sea estructurado de una manera tal que permita que el conocimiento relevante a un evento específico pueda ser extraído cuando se requiera.

La estructura de información de un sistema inteligente basado en conocimiento podemos dividirla en tres niveles : datos físicos, base de conocimiento y estructura de control.

El primer paso fue diseñar un Sistema de Información espacial que pudiera efectuar una serie de operaciones sobre muchos archivos de datos que describen las características de los recursos de la tierra del Estado de Louisiana. Estos archivos se obtuvieron de la digitalización de diversos mapas tales como : grosor de la veta de lignito, profundidad, propiedad de la tierra, red de caminos, uso de la tierra, división política, etc. Esta información es la que constituye los "datos físicos".

El sistema espacial de información es capaz de realizar una amplia gama de operaciones sobre los distintos archivos de datos espaciales por medio de comandos primitivos que reconoce a través de una sintaxis diseñada especialmente.

A continuación se sitúa otro sistema que contiene la "base conocimiento" y que incluye el conocimiento conceptual requerido para decidir la tarea específica que se está desarrollando.

Mientras por un lado nos hemos basado en un considerable número de conceptos obtenidos de expertos humanos relacionados con conocimiento de una disciplina específica, en este caso Geología Económica, también utilizamos inferencias selectivas para descubrir la naturaleza generalizada de ciertos conjuntos de eventos, para describir estos eventos en términos de atributos generales.

La "estructura de control" de nuestro sistema es de la forma reducción del problema, donde la aplicación de reglas de producción a varios niveles ya sea, dirigen hacia un conjunto de reglas o eliminan otro conjunto de reglas que puedan ser aplicadas.

El problema es, entonces, determinar cuáles ocurrencias del recurso mineral son : (1) económicamente factibles, basado en una razón de recuperación mínima, (2) ambientalmente factibles basado en múltiples factores ambientales, y (3) lo suficientemente grande para garantizar la explotación.

Una sesión típica se inicia con una introducción que el sistema hace al usuario donde además lo guía a identificar el área de interés. Una vez que el área está identificada, se investiga la factibilidad económica usando grosor de veta y profundidad de ésta o razón de recuperación, que puede introducir el usuario o usar el criterio de la base de conocimiento del sistema.

A continuación se investiga la factibilidad ambiental donde se analizan múltiples factores como uso de la tierra, donde se usan diferentes ponderaciones en caso de ser potencial recurso de agua o agrícola, distancia a vías camineras, propiedad de la tierra, etc. Inmediatamente después se ejecuta el módulo que investiga si las entidades espaciales elegidas previamente conforman conjuntos contiguos lo suficientemente grandes que garanticen la explotación.

Finalmente, la salida se obtiene en un terminal RAMTEK de alta resolución, en la forma de áreas coloreadas representando los diversos niveles de conveniencia de explotación.

Si el usuario que está participando en la sesión es un experto en geología, el sistema le permite modificar temporalmente su base de conocimiento para usar factores de juicio del experto. Asimismo, el sistema reporta en cualquier momento el por qué se ha tomado cada decisión, es decir está capacitado para explicar su línea de razonamiento en un ambiente altamente interactivo, lo que le da su característica de transparencia.

CONCLUSIONES

Este trabajo demostró su capacidad en la aplicación descrita y servirá como modelo para implementar otro tipo de aplicaciones similares en otras áreas. Estamos trabajando en incorporarle al sistema la capacidad de modificar su propia base de conocimiento, lo que le agregará la característica de "aprendizaje".

BIBLIOGRAFIA

1. Buchman, Bruce y Duda, Richard : Principles of Ruled-Based Expert Systems Fairchild Technical Report N°626, Fair Technical Report N°9 August 1982. pp. 11-12, 1982.

2. Iyengar S. Sitarama : Knowledge Based Systems with reference to medical applications Paper in Proceedings of IEEE-COMPCON 83 pp. 2-3, 1982.

3. International Business Machines Corporation : Interactive System Production Facility, Dialog Managemet Servece Publication Number Sc 34-2088-1, Second Edition. (May, 1983)., 1983.

4. Webster, R. y Miner, Leslie : Expert Systems, programming problem solving. Technology, January-February 1982, pp. 62., 1982.

5. Miller, Stephan y Iyengar, S. Sitharama : "Representation of regions of map data for comparison and retrieval Proceedings of the Computer Vision and Pattern Recognition Conference, pp. 2-3, 1983.

C.M. DAL SASSO-FREITAS*

SUMÁRIO

O trabalho propõe a utilização de diagramas de transição de estados no processo de modelagem de diálogos gráficos, dentro de uma metodologia já conhecida. Modificações nos símbolos utilizados como estados são sugeridas para aumentar a potencialidade de representação dos diagramas.

ABSTRACT

This paper suggests the use of state transition diagrams to represent graphical dialogues when applying a well-known design methodology. Modifications in the symbols used as states are suggested to increase the representational capability of the diagrams.

*Mestre em Ciência da Computação (UFRGS, 1982); professora assistente do Departamento de Informática (CPD, UFRGS) e do Curso de Pós-Graduação em Ciência da Computação (UFRGS). Endereço: UFRGS-CPGCC, Caixa Postal 1501, 90.000 - Porto Alegre, RS, Brasil.

INTRODUÇÃO

Em trabalho anterior /1/ discutiu-se o projeto de diálogos gráficos tendo como base o uso de uma metodologia proposta em /2/, estendida em um nível conforme /3/. A metodologia não menciona, no entanto, modelos formais para representação do diálogo. Neste sentido, o estudo prosseguiu /4/ e seus resultados preliminares são aqui apresentados.

O uso de métodos formais de representação de diálogos surge com frequência na literatura /5-9/, tanto para dar maior confiabilidade ao projeto como para subsidiar processadores universais de diálogo /6, 9, 10, 11/.

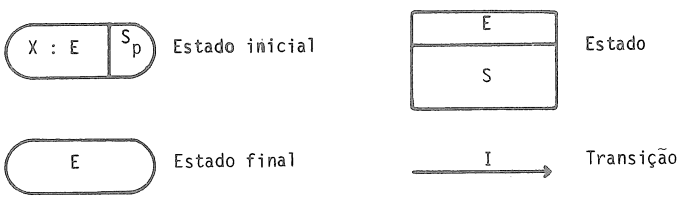
Após a análise de diversas formas de representação /4/, a saber, diagramas de transição de estados, árvores de escolha, gramáticas formais, redes e células de diálogo, optou-se por utilizar diagramas de transição de estados, com algumas modificações.

Um diagrama de transição de estados representa uma máquina de estados que é definida como uma quíntupla (A,S,Z,f,z), onde A é um conjunto finito de símbolos de entrada, Z é um conjunto de símbolos de saída, S é um conjunto finito de estados, f é uma função próximo - estado $S \times A \rightarrow S$ e z é uma função de saída $S \times A \rightarrow Z$. No diagrama, os estados são representados por círculos (com o nome do estado inscrito) e as transições são arestas com inscrições representando símbolo de entrada e símbolo de saída.

DIAGRAMA DE TRANSIÇÃO DE ESTADOS MODIFICADO

Para representar um diálogo através de um diagrama de transição de estados associa-se: (a) ao conjunto S, o conjunto de estados válidos do sistema, (b) ao conjunto A, o conjunto de entradas passíveis de serem fornecidas pelo usuário - requisição de funções e parâmetros -, (c) ao conjunto Z, o conjunto de mensagens e imagens exibidas pelo sistema e (d) às funções f e z, funções do sistema.

Através de modificações nos símbolos utilizados nos diagramas de transição de estados obtém-se maior representatividade. Os símbolos modificados são apresentados abaixo:



No estado inicial, X é o nome do diagrama, E é o nome do estado e S_p é o conjunto de imagens e mensagens presentes durante todos os estados do diagrama.

Em todos os demais estados do diagrama, S é o conjunto de mensagens e imagens que permanecem durante o estado em que estão inscritas.

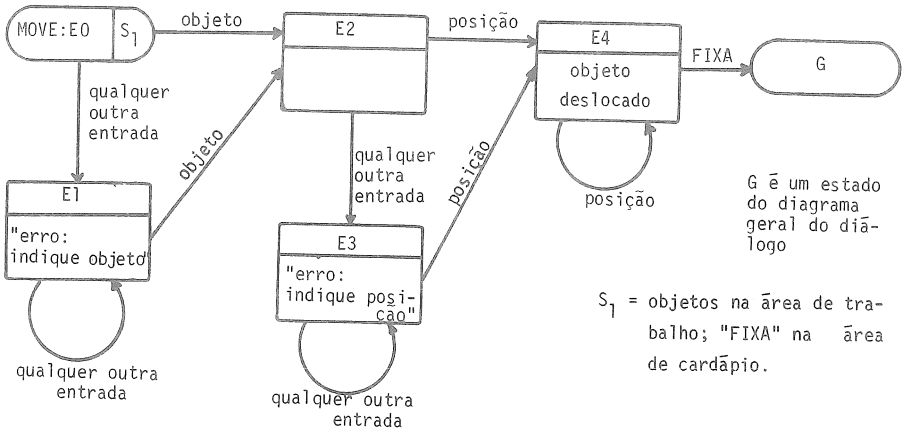
Na transição, I representa uma entrada do usuário. Tanto S como I podem ser definidas à parte e referidas, no diagrama, por um identificador. Qualquer estado pode ser detalhado bastando que se construa um diagrama separado onde seu nome aparece no estado inicial. O estado final de um diagrama, pode, portanto, aparecer em outro diagrama.

PROCESSO DE MODELAGEM DE DIÁLOGOS GRÁFICOS

A metodologia de projeto mencionada anteriormente constitui-se de seis etapas: análise da tarefa, especificação do modelo conceitual da aplicação, projeto da semântica do diálogo, projeto da sintaxe, projeto da léxica e especificações de implementação.

Diagramas de transição de estados são introduzidos no projeto da sintaxe e, a partir, daí são trabalhados em função das decisões sobre a natureza (léxica) das unidades sintáticas e sobre a implementação de cada uma delas (técnicas interativas e dispositivos adotados).

Tomemos como exemplo uma operação simples de movimentação de um objeto exibido na tela. Após a escolha da operação MOVIMENTO, o sistema passa a esperar a indicação do objeto e, depois, as posições para onde ele deve ser deslocado, até ser fixado.



Dependendo das técnicas interativas adotadas para implementar a entrada dos parâmetros, o estado E2, por exemplo, pode ter como saída um lembrete para entrada da posição. Por isso, o diagrama pode ser modificado, refletindo alterações na sintaxe, devidas a decisões de implementação. Sabe-se que um dos elementos mais importantes no projeto de interfaces gráficas é a escolha das técnicas interativas e a ordenação no tempo do uso dos diferentes dispositivos /12/.

CONCLUSÕES

O método está sendo testado em diversas aplicações experimentais que serão parcialmente implementadas. Uma opção de continuidade do presente trabalho pode ser a implementação de um processador que, com base numa descrição tabular dos diagramas, como a sugerida em /5/, conduza o sistema automaticamente, ou seja, acione as rotinas aplicativas intrínsecas a cada estado. Deste modo, não haveria necessidade de programar a interface, apenas de descrevê-la para o processador.

REFERÊNCIAS

/1/ DAL SASSO-FREITAS, C. M. Considerações sobre o projeto de diálogos homem-máquina e o projeto de modelos em aplicações gráficas interativas. In: CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA, 10., Vina Del Mar, abril 1984. Anais. Chile, 1984. p.263-5.

/2/ FOLEY, J.D. & VAN DAM, A. Fundamentals of interactive computer graphics. Reading, Addison-Wesley, 1982.

/3/ BUXTON, W. Lexical and pragmatic considerations of input structures. Computer Graphics, 17(1):31-7, Jan. 1983.

/4/ DAL SASSO-FREITAS, C. M. Modelagem de diálogos gráficos. Porto Alegre, PGCC da UFRGS. (RP a ser publicado)

/5/ BARNES, B.H. An automata theoretic approach to interactive computer graphics command languages. In: YEH, R.T., ed. Applied Computation Theory. Englewood Cliffs, Prentice-Hall, 1976. p.563-81.

/6/ BLACK, J.L. A general purpose dialogue processor. In: NATIONAL COMPUTER CONFERENCE, Dallas, 1977. Proceedings. Montvale, AFIPS, 1977. p.397-408.

/7/ DENERT, E. Specification and design of dialogue systems with state diagrams. In: INTERNATIONAL COMPUTING SYMPOSIUM, Liege, Apr. 1977. Proceedings. Amsterdam, Nort-Holland, 1977. p.417-24.

/8/ REISNER, P. Formal grammar and human factors design of an interactive graphics system. IEEE Trans. on Soft. Engineering, SE-7(2):229-40, Mar. 1981.

/9/ SILVA, M.V. & MAGALHÃES, L.P. Processadores universais de diálogo: formas de representação do diálogo. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO,4., Viçosa, 1984. Anais. Viçosa, SBC, 1984. p.15-29.

/10/ BUXTON, W.; LAMB, M.R.; SHERMAN, D.; SMITH, K.C. Towards a Comprehensive user interface Management System. Computer Graphics, 17(3):35-41, July 1983.

/11/ OLSEN, Jr., D.R. & DEMPSEY, E.P. SYNGRAPH: A Graphical user interface generator. Computer Graphics, 17(3):43-9, July 1983.

/12/ FOLEY, J.D.; WALLACE, V.L.; CHAN, P. The human factors of computer graphics interaction techniques. IEEE Computer Graphics and Appl., 4(11):13-48, Nov. 1984.

A.E. COSTA PEREIRA *
M.C. MONARD **
R.ALLAN ***

SUMARIO

A Programação Lógica possui características que tornam mais fácil e rápido o desenvolvimento de programas, mas a maioria dos interpretadores e compiladores PROLOG não oferecem as ferramentas necessárias para o processamento de dados numéricos e para o desenvolvimento de pacotes gráficos. Nesta comunicação, mostramos algumas extensões que estamos implementando num interpretador PROLOG a fim de capacitá-lo para este tipo de aplicações.

ABSTRACT

Logic Programming has features which make the development of programs easy and fast but most of PROLOG interpreters and compilers do not provide the necessary tools for processing numerical data and for developing graphic packages. In this communication we show some extensions we are implementing in a PROLOG interpreter to make it suitable for this sort of applications.

* Bacharel em Física (IF-USP,1971); Eng. Eletrônico (EP-USP,1971); MSc(INPE-1975); PHd (Univ. Cornell-USA,1979); Livre Docente (UNESP-Rio Claro, 1983). Inteligência Artificial, Programação Lógica, Sistemas Especialistas.

** MSc(Univ. Southampton-Ingl.,1969); Dr. Informática (PUC-RJ,1980). Estrutura de Dados e Algoritmos, Programação Lógica
USP-ICMSC. Depto. Computação. Av. Dr. Carlos Botelho, 1465. CEP 13560. São Carlos - SP.

*** Bacharel em Computação (USP-São Carlos, 1983); Atualmente Aluno de Mestrado em Ciência da Computação no INPE. Inteligência Artificial, Programação Lógica.

INTRODUÇÃO

A linguagem de programação PROLOG é conhecida como uma ferramenta muito poderosa para computação simbólica. No que se refere a computação numérica e gráfica, apresenta-se ineficiente devido ao fato que a maioria dos interpretadores e compiladores PROLOG não possuem as ferramentas necessárias para processar dados numéricos e desenvolver pacotes gráficos.

O objetivo de nosso trabalho é a implementação de um interpretador PROLOG com tais ferramentas a fim de facilitar a programação e rapidez de resposta nas áreas referidas acima.

DESCRIÇÃO

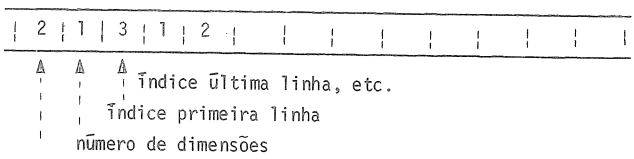
A implementação do interpretador é feita usando a técnica de "structure sharing" [1,2,3] com quatro pilhas. Pilha de substituições, pilha de falha, pilha de variáveis e pilha de hipótese. As variáveis são divididas em locais e globais [2,4]. Os detalhes da implementação e estrutura de dados são descritos em [5].

No caso do problema de computação numérica, é necessário uma representação interna simples, de rápido acesso e que ocupe pouco espaço. Decidimos implementar uma estrutura chamada "estrutura tabela" que deve ser especificamente declarada.

Por exemplo, a declaração

```
def-tab V(1..3,1..2).
```

gera a seguinte estrutura



Nesta implementação, cláusulas e lementos de uma tabela são tratados da mesma forma. Do ponto de vista do usuário, as cláusulas

```
V(1 1 5); V(1 2 3.7); V(2 2 -8.0E-4); ... V(3 1 -57)
```

são equivalentes a declarar V como tabela e posteriormente preencher os elementos de V com os valores correspondentes.

Porém, tabelas usam menos espaço que cláusulas. Além disso, se uma cláusula unifica com a hipótese corrente, a próxima cláusula com a mesma relação é colocada na pilha de falhas para ser usada quando fizermos "backtracking", mas no caso de unificação com um elemento da "estrutura tabela", o endereço da tabela e o índice do próximo elemento são guardados na pilha de falhas.

Esta forma uniforme de tratar tabelas e cláusulas permite o uso de "backtracking" para percorrer os elementos de uma tabela e construir facilmente programas para manipular es-

ta estrutura.

No caso do problema de computação gráfica, o uso do "bracktracking" de PROLOG torna-se muito interessante. Ao efetuar a expansão na árvore de prova, vai-se desenhando a figura e, no caso de acontecer "backtracking", os pontos traçados podem ser apagados e uma outra figura é tentada.

Não é difícil implementar "turtle graphics" usando o "backtracking" de PROLOG. Para isto é suficiente armazenar na pilha informações de cada segmento de linha que é desenhado e armazenar os valores correntes das variáveis. Quando acontecer "backtracking" estas informações são usadas para apagar estes segmentos de linha.

Achamos que isto é bastante prático para o pessoal ligado a CAI e CAD. Aproveita-se o não determinismo para gerar figuras até chegar a alguma que satisfaz as condições necessárias.

CONCLUSÕES

Com estas duas extensões acreditamos que a programação lógica para processos numéricos e gráficos possa ser facilitada diminuindo o tempo de desenvolvimento de programas e o tempo de execução dos mesmos.

REFERÊNCIAS

1 - Boyer,R.S.; Moore,J.S - The Sharing of Structure in Theorem Proving Programs. Machine Intelligence 7. Edimburgh University Press. 1972.

2 - Bruynooghe,M. - The Memory Management of PROLOG Implementations. Logic Programming (EDs. Clark K.L.; Tarnuld S.A.). Academic Press, New York, NY. pp. 84-98, 1982.

3 - Van Emden,M.H. - An Algorithm for Implementing PROLOG Programs. University of Waterloo, 1980.

4 - Warren,D.H.D. - Implementing PROLOG-Compiling Logic Programs. DAI Research Report Nº 39-40. University of Edinburh, 1977.

5 - Grupo Computação Lógica: Detalhes da Estrutura de Dados e Implementação de Um Interpretador PROLOG. Documento Interno ICMSC-USP, 1985.

ESTAGIO ATUAL DO DESENVOLVIMENTO DE UM INTERPRETADOR PROLOG EM PASCAL SEQUENCIAL

A.L.B.DIETERICH *

SUMARIO

Este trabalho descreve a implementação de um interpretador PROLOG em Pascal Seqüêncial (de Per Brinch Hansen) no CPGCC-UFRGS. O Interpretador é composto de 4 módulos: definição das cláusulas, execução do programa, comunicação interativa com o usuário e gerenciamento da área de trabalho.

ABSTRACT

This paper describes the implementation of a PROLOG interpreter, written in Sequencial Pascal (by Per Brinch Hansen) at CPGCC-UFRGS. The Interpreter is composed of four modules which are: clauses definition, program execution, interative comunication with the user and working area management.

* Tecnólogo em Processamento de Dados (UFRGS,1985) em programa de Mestrado em Ciências da Computação-UFGRS, inteligência artificial, banco de dados, Av. Iguaçú 659 -Porto Alegre - RS - 90000.

INTRODUÇÃO

Um programa em PROLOG é constituido de fatos, regras e perguntas. Os fatos e as regras permanecem armazenados, em um arquivo, como programa objeto. Este programa é executado cada vez que uma pergunta é feita. A execução de um programa significa consultar este arquivo que contém os fatos e as regras (de agora em diante referenciados como cláusulas).

A sintaxe adotada foi a sintaxe de Edinburg, porém com listas através do operador "ponto".

REPRESENTAÇÃO INTERNA

Para a representação interna de um programa Prolog utiliza-se uma árvore binária, como estrutura principal, e um dicionário, que é referenciado por esta. A árvore é utilizada para implementar uma estrutura hierárquica de diversos níveis, cada nível correspondendo a uma subestrutura específica, conforme descrito a seguir.

O nível Ø organiza o programa em pacotes. No nível 1 existe uma lista de cláusulas, contendo um nodo para cada cláusula do pacote. No nível 2, caso a cláusula seja uma regra, estarão encadeadas as condições desta. Os níveis de 3 a n-1 representam o aninhamento das funções que constituem termos estruturados. O último nível (n) representa finalmente o termo, através de apontadores para o dicionário que representa os átomos do programa.

DEFINIÇÃO DAS CLÁUSULAS

Este módulo faz a tradução das cláusulas em termos da representação interna. Isto é feito através da análise sintática descendente recursiva que se encarrega, ao mesmo tempo, de disparar rotinas geradoras de código e de, através de rotina específica, passar ao próximo item léxico.

EXECUÇÃO DO PROGRAMA

O módulo execução do programa, ainda não implementado, é responsável pela resposta de perguntas através da consulta à representação interna do programa. Para a análise da pergunta serão utilizadas as rotinas do módulo definição das cláusulas, já implementadas. Para a execução, propriamente dita, faltam basicamente as rotinas de controle e de unificação, que estão em andamento.

COMUNICAÇÃO INTERATIVA COM O USUÁRIO

Os módulos descritos anteriormente encarregam-se do processamento das informações, usando arquivos como forma principal de entrada de dados. Cabe-

rá a este módulo tornar o sistema conversacional, facilitando a entrada de dados.

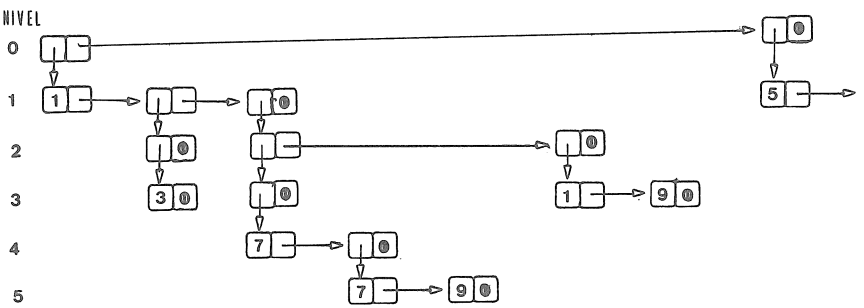
GERENCIAMENTO DA ÁREA DE TRABALHO

Este módulo encarregar-se-á de ocupar de forma mais otimizada a área de memória principal disponível, implementando uma memória virtual.

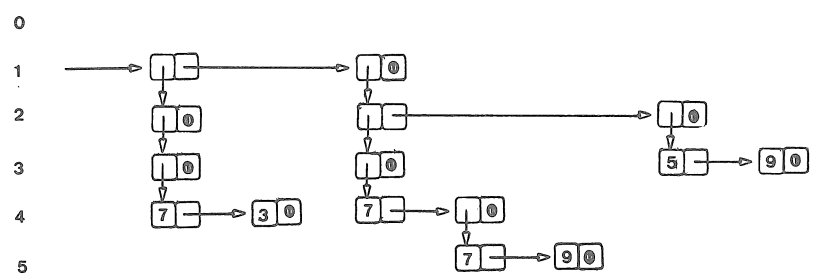
EXEMPLO

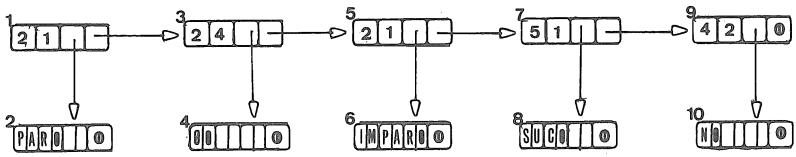
```
par (Ø).
impar (suc (Ø)).
par (suc(suc(N))):- par (N).
impar (suc (suc(N))):- impar (N).
```

ESTRUTURA PRINCIPAL



(CONTINUAÇÃO)





Já foram feitas a definição da sintaxe e da representação interna do programa PROLOG, e a implementação do Módulo definição das cláusulas,estando agora em estudo a implementação do Módulo execução do programa.

BIBLIOGRAFIA

BRINCH HANSEN, Per. The Architeture of Concurrent Programs. New Jersey, Prentice-Hall, Inc., 1977.

CLOCKSIN, William F. & MELLISH, Christopher S. Programming in Prolog. Berlin, Heidelberg, Springer-Verlag, 1981.

CLOCKSIN, William F. & YOUNG, J. Introduction to Prolog, a "fifth-generation" language. Computerworld, 17 (31), Aug.1,1983.

JENSEN, Kathleen & WIRTH, Nicklaus. Pascal, User Manual and Report. Berlin, Springer-Verlag, 1974.

LIBERMAN, Miriam & BIDART, João Carlos Remedi. Introdução a Estrutura de Dados. Porto Alegre, UFGRS, CPD, 1983.

PAGE, E.S. & WILSON L.B. Information Representation and Manipulation in a Computer. Cambridge, University Press. 2nd Ed, 1978.

S.R.M. Pellegrino*
J.A.G. Pereira**

SUMÁRIO

Este trabalho apresenta um estudo preliminar para a obtenção automática de isóbaras e isotermas de superfície. Este tipo de traçado tem como objetivo auxiliar o analista de meteorologia na análise de cartas sinóticas. Desenvolveram-se algumas estratégias para melhorar a apresentação das isolinhas bem como facilitar sua utilização.

ABSTRACT

This work presents a preliminary study towards the automatic production of isobars and isotherms surface. This plotting aims at helping the meteorologist in the analysis of sinoptic charts. Some strategies were developed to improve the representation of the isolines, as well as make its utilization easier.

* Matemática (PUC/SP, 1980), Física (PUC/SP, 1981). Mestre em Computação Aplicada (INPE, 1985). Áreas de interesse: análise numérica, computação gráfica. Atualmente Assistente de Pesquisas no Instituto de Pesquisas Espaciais (INPE) - Av. dos Astronautas, 1758 - Caixa Postal 515, 12200 - São José dos Campos - SP.

** Físico (PUC/RJ, 1966). Mestre em Computação Aplicada (INPE, 1974), Doutor em Computação Aplicada (INPE, 1982). Áreas de interesse: processamento de imagens, computação gráfica, métodos numéricos, análise de classes. Atualmente Pesquisador Associado no Instituto de Pesquisas Espaciais (INPE) - Av. dos Astronautas, 1758 - Caixa Postal 515, 12200 - São José dos Campos - SP.

RESUMO ESTENDIDO

Dando continuidade ao trabalho de traçado automático de cartas sinóticas [2] desenvolvido no INPE, optou-se por aprimorar a ferramenta já obtida no sentido de facilitar a tarefa do analista de meteorologia para obter a análise da carta sinótica.

Partindo de dados reais de temperatura e pressão, que se encontram distribuídos de forma irregular no plano do papel, escolheu-se, nesta primeira tentativa, a utilização de um interpolador que recebe como dados de entrada pontos distribuídos desta forma e fornece, como dados de saída, uma grade regular que pode ser utilizada como base para o traçado de isolinhas.

O método de Interpolação Bivariável para pontos distribuídos irregularmente [1] foi escolhido por apresentar a característica acima mencionada e ainda por gerar uma superfície de interpolação suave; ser rápido quando comparado a outros interpoladores; e utilizar elementos finitos que permitem maior adaptação em regiões que exigem maiores detalhes.

Para o traçado efetivo das isolinhas (isóbaras e isotermas) buscou-se ainda na literatura um método que fosse capaz de executar a tarefa.

O método utilizado [3] tem como entrada a grade regular gerada pelo interpolador [1]. Este método de obtenção de isolinhas faz a busca em uma grade retangular, que é considerada a região do traçado, segundo uma espiral que indica qual a próxima cela a ser visitada. Por este processo não há visitas em celas que já foram pesquisadas, forçando assim um decremento no tempo de processamento.

Neste trabalho a região que contém os dados é aquela limitada pelo contorno do mapa da América do Sul, o que causa o primeiro problema, ou seja, os valores da grade regular interpolados fora da região onde se concentram os dados não são de boa qualidade. Portanto, de acordo com o esperado, o traçado das isolinhas nesta região não corresponde à realidade do fenômeno e, as curvas obtidas convergem para um mesmo ponto.

A idéia foi criar uma máscara numérica que não alterasse a grade gerada pelo interpolador e que não modificasse o processo de busca das rotinas de obtenção de isolinhas. A função desta máscara é então de permitir que as isolinhas sejam traçadas apenas na superfície limitada pelo contorno da América do Sul. Esta modificação resultou num traçado mais aceitável.

Convém ressaltar que o método adaptou-se bem às isotermas; para as isóbaras o traçado não representa o fenômeno físico. Esta disparidade em relação às isóbaras ainda não está bem entendida, mas pode-se seguramente concluir que um dos fatores que contribuem para este desvio são os dados de geopotencial enviados por algumas estações no lugar do valor da pressão. Neste trabalho não é levado em consideração o relevo da região o qual, segundo os meteorologistas, é um fator importante para a obtenção das isóbaras.

Quanto à parte numérica, acredita-se que esta discrepância encontrada no traçado possa ser gerada devido ao fato de a relação entre os valores de pressão ser muito pequena, o que pode ocasionar a geração de polinômios com grande oscilação dentro do intervalo pesquisado.

Para tornar mais razoável o acompanhamento de uma linha de contorno, mudou-se a forma de identificá-las. Ao invés de inserir uma marca na linha, no ponto inicial do seu traçado, onde o usuário após identificar o símbolo desenhado tem de percorrer uma tabela para obter o valor da cota, substitui-se este símbolo pelo próprio valor da cota. Teve-se ainda a preocupação de interromper a linha em trechos variados, inserindo aí o valor da cota e facilitando a identificação da linha.

Por questões estéticas, o valor de identificação da linha de contorno deve acompanhar a inclinação da curva. Para isto, foi necessário mais uma vez alterar a rotina, desta vez com o propósito de armazenar a posição em relação aos eixos cartesianos da carta, nos quais a pena da "plotter" se encontrava na posição anterior à atual, para saber qual ângulo a linha forma em relação ao eixo-base.

Acredita-se que, com a experiência obtida neste trabalho, é possível superar os problemas encontrados e obter um traçado satisfatório para as isotermas e sobretudo para as isóbaras.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] AKIMA, H. A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregular Distributed Data Points. ACM Transactions on Mathematical Software, 4(2):148-159, June, 1978.

[2] PELLEGRINO, S.R.M.; PEREIRA, J.A.G. Traçado Automático de Cartas Sinóticas, INPE-3146-RPE/456, Junho 1984.

[3] SNYDER, W.V. Contour Ploting [16]. ACM Transactions on Mathematical Software, 4(3):290-294, Sept. 1978.

M. ROISENBERG*
L. A. CARVALHO**

SUMARIO

O objetivo desse trabalho é implementar a linguagem GRAFCET como gerenciadora, em um nível mais alto, de um programa tradicionalmente escrito em linguagem de diagrama de relês. Desta forma é descrita a estrutura de dados capaz de representar a linguagem e o processamento efetuado, considerando fatores como velocidade de execução e código gerado, atributos importantes na área de controle de processos em tempo real.

ABSTRACT

The aim of this paper is to describe the implementation of the GRAFCET language as a high level manager of control program traditionaly write' with relay ladder diagram language. In this way, we describe the data structure able to represent the language and how the proceedings are ' doing, considering factors as execution speed and code generation, ' important qualities in real time control process area.

* Tecnólogo em processamento de dados (UFRGS, 1984) ; mestrando em ciência da computação (PGCC, UFRGS) ; P&D Altus Sistemas de Informática Ltda ;controle de processos, sistemas operacionais.

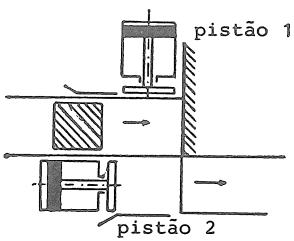
**Engenheiro eletrônico (UFRGS, 1984) ; mestrando em ciência da computação (PGCC, UFRGS) ; P&D Altus Sistemas de Informática Ltda ; controle de processos, microeletrônica.

O GRAFCET (Grafos de Comando Etapa-Transição) é uma ferramenta baseada na teoria de Redes de Petri utilizada para descrição de sistemas lógicos em geral e particularmente de processos de automação industrial. Ela se compõe de três elementos básicos: etapas; transições e arcos. Etapas ou lugares são elementos aos quais estão associadas as ações do processo. Transições são elementos aos quais estão associados eventos. Os Arcos são elementos que ligam lugares a transições e essas a novos lugares. Num grafo, um lugar pode estar ativado ou não em um determinado instante, sendo a ativação representada por uma marca.

Como o GRAFCET em certas estruturas necessárias em controle industrial se mostra pouco eficiente, optou-se por torna-lo o gerenciador das funções de controle, essas, implementadas em linguagem de relê.

Com o intuito de tornar executável a descrição de um processo em GRAFCET procurou-se uma estrutura de dados capaz de representar genericamente e de forma unívoca a linguagem.

O exemplo da figura 1 mostra um processo simples de transferência de peças e sua descrição em GRAFCET.



LUGAR
1:espere
2:avance pistão 1
3:avance pistão 2
4:recue pistão 2
5:pare pistão 2
6:recue pistão 1
7:pare pistão 1

TRANSIÇÃO
1:peça na frente do pistão 1
2:peça na frente do pistão 2
3:peça saindo
4:pistão 2 recolhido
5:pistão 1 recolhido
6: "1"

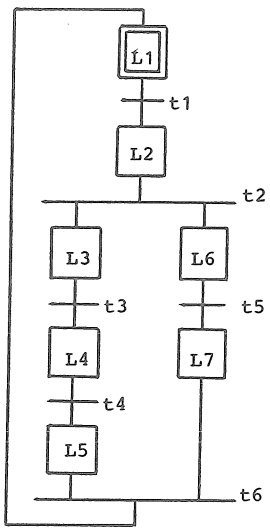


Figura 1. Descrição de um processo em GRAFCET

ARQUITETURA DO SISTEMA

O sistema é composto de um controlador programável AL-1000 fabricado pela Altus Sistemas de Informática, um terminal de programação AL 1800 do mesmo fabricante e um programador GRAFCET implementado num MAXXI da Polimax.

O controlador é o elemento responsável pela execução da lógica de controle, sendo constituído por uma UCP e interfaces de E/S. Esta UCP possui um microprocessador 8031 e 8k de memórias para armazenamento das tabelas do GRAFCET mais 16K para armazenamento das lógicas de relé. No TP AL1800 é feita a entrada edição e montagem da parte referente a lógicas de relé. Essas mesmas tarefas no que se refere ao GRAFCET são efetuadas no microcomputador MAXXI.

Os programadores, MAXXI e AL1800, comunicam-se com o Controlador através de um canal serial a 9600 baud.

ESTRUTURA DE DADOS

Tendo em vista o espaço de memória disponível limitou-se a 100 o número de lugares, sendo permitido o mesmo número de transições.

A estrutura é organizada em 3 tabelas como segue:
- Tabela de lugares (TL) com no máximo 1,5K bytes.
- Tabela de Transições (TT) com no máximo 2,5K bytes.
- Tabela de Lugares Ativos (TLA) com no máximo 15 lugares ativos em determinado instante.

A TL possui três campos de informação: as transições que necessitam deste lugar ativo para ocorrerem, o número de marcas neste lugar e o endereço inicial das lógicas em diagrama de relé que devem ser executadas em virtude do lugar estar ativo. A TT também possui três campos: os lugares que devem possuir marcação para que a transição ocorra, os lugares para onde é transferida a marcação quando a transição ocorrer e os eventos necessários para sua ocorrência. A TLA contém os apontadores que assinalam, dentro da TL, os lugares com marcação e portanto sendo executados. As tabelas que representam o processo visto na fig. 1 estão apresentadas na figura 2.

Além disso convêm ressaltar que a parte referente a ação dos lugares, ou seja, aquela implementada em lógicas de relê, é executada diretamente pelo controlador.

ALGORITIMO DE EXECUÇAO

A execução do algorîtimo de controle se constitui em um laço núcleo onde além da atualização dos pontos de entrada e saîda do processo controlado é executado o programa de aplicação através da manipulação da estrutura de dados e da ação referente aos lugares. Consultando a TLA,o interpretador do GRAFCET sabe quais são os lugares ativos, buscando em TL o endereço inicial das lógicas que devem ser executadas para cada lugar ativo. O retorno ao interpretador é efetuado ao final da última lógica executada, por uma instrução Retorno. Após isso, são analisadas as transições dependentes daquele lugar ativo. Esta anâlise consiste na verificação se os lugares de origem daquela transição estão ativos e se os eventos necessários ocorreram. Um evento é o valor lógico que deve ter um relê do sistema para que a transição ocorra. Caso o evento tenha ocorrido, os lugares destino da transição são colocados na TLA e recebem marcas, e o lugar origem, se tiver seu número de marcas igual a zero é retirado da TLA. Caso a transição não tenha ocorrido, então o próximo lugar é executado.

CONCLUSAO

A linguagem GRAFCET possibilita a programação de sistemas de controle em um nîvel mais elevado do que os até hoje existentes em controladores programáveis nacionais, pois possibilita a visualização do processo como uma máquina de estados sendo possîvel a descrição de ações paralelas e sequenciais. Além disso representa uma nova ferramenta, favorecendo a implementação do programa descritor do processo de forma "top-down", e o uso de algorîtimos de anâlise de Redes de Petri, hoje jâ conhecidos, para deteção de "dead lock", estados atingîveis, etc.

TABELA DE LUGARES

	TRANSIÇÕES	NMARC	END.LÓGICA
L1	t1	1	30
L2	t2	0	57
L3	t3	0	170
L4	t4	0	193
L5	t6	0	305
L6	t5	0	350
L7	t6	0	400
	10 bytes	1 byte	2 bytes

TABELA DE LUGARES ATIVOS

1	L1
0	
0	
0	

15 bytes

ativo — nº do lugar

TABELA DE TRANSIÇÕES

	LUGARES ORIGEM	LUGARES DESTINO	EVENTO
t1	L1	L2	A001
t2	L2	L3, L6	A370
t3	L3	L4	A220
t4	L4	L5	A010
t5	L6	L7	A162
t6	L5, L7	L1	A200
	10 bytes	10 bytes	1 byte

Figura 2. Estruturas de dados utilizadas

BIBLIOGRAFIA

(1) BOSSY, J. C.; BRARD, P.; FAUGÈRE, P.; MERLAUD, C.; "Le Grafcet, sa pratique et ses applications"; Educalivre, Paris, 1979.

(2) RILLO, M.; COSTA RILLO, A. H. R.; CIRELLI, S.; ZILINSKAS, W.; "Controlador Programável por Grafos de Comando Etapa-Transição (GRAFCET)", in Anais do 1º Congresso Nacional de Automação Industrial, São Paulo, Julho 1983 (246-252).

(3) ROISENBERG, M.; "Expansão do Sistema Operacional do Controlador Programável AL-1000"; CFTPD-UFRGS, Porto Alegre, 1984.

M.A. CASANOVA*, L. TUCHERMAN**, A.L. FURTADO***

SUMÁRIO

Um método para projeto de banco de dados, baseado no conceito de módulos, é inicialmente descrito. Este método incorpora tanto uma estratégia para garantir restrições de integridade quanto uma tática para organizar grandes conjuntos de estruturas, restrições de integridade e operações de banco de dados. Em seguida, o protótipo de um sistema especialista que auxilia a criação de esquemas modulares de banco de dados é apresentado. Este protótipo, escrito em uma extensão de micro-PROLOG, é parte de uma ferramenta mais abrangente destinada a facilitar o desenvolvimento e manutenção de esquemas conceituais modulares.

ABSTRACT

A database design method, based on the concept of module, is first described. The method incorporates both a strategy for enforcing integrity constraints and a tactic for organizing large sets of database structures, integrity constraints and operations. A prototype expert system to help create modular database schemas is then presented. The prototype, written in an extension of micro-PROLOG, is part of a more comprehensive design tool to help develop and maintain modular conceptual schemas.

* Doutor em Matemática Aplicada (Harvard Univ., 1979); teoria de bancos de dados, sistemas de gerência de bancos de dados distribuídos, projeto e análise de algorítmos concorrentes; Centro Científico de Brasília, IBM do Brasil - Caixa Postal 853 - Brasília, DF - CEP 70.000

** Mestre em Ciência da Computação (PUC/RJ, 1983); projeto de bancos de dados, sistemas de gerência de bancos de dados distribuídos, sistemas especialistas; Instituto Latino Americano de Pesquisa em Sistemas, IBM do Brasil - Caixa Postal 1830 - Rio de Janeiro, RJ - CEP 20.071

*** Doutor em Ciência da Computação (Univ. Toronto, 1974); projeto de bancos de dados, tipos abstratos de dados, sistemas especialistas; Depto. de Informática, PUC/RJ - Rua Marquês de S. Vicente, 209 - Rio de Janeiro, RJ - CEP 22.453

INTRODUÇÃO

Discutiremos neste trabalho uma ferramenta de software que auxilia o administrador de banco de dados (ABD) a especificar e manter esquemas de banco de dados seguindo uma disciplina modular. Apenas a fase inicial da implementação da ferramenta, consistindo de um dicionário para armazenar a descrição de esquemas conceituais modulares e de facilidades para adicionar novos módulos a um esquema existente, será apresentada. A especificação completa da ferramenta é descrita em [TFC].

A ferramenta incorpora conhecimento sobre um método de projeto de banco de dados [TCF] que oferece descrições estruturadas das noções tradicionais de esquema conceitual e esquema externo. Esquemas relacionais, restrições de integridade e operações são agrupadas em módulos [Pa,LZ,ZLT] e apresentadas de maneira ordenada e estruturada, o que permite aumentar o entendimento sobre o banco de dados. O método também exige que as relações de um módulo M sejam atualizadas somente pelas operações definidas em M, o que corresponde à noção usual de encapsulamento [LZ]. Portanto, se as operações de cada módulo M preservarem consistência com relação às restrições de integridade de M, o método oferece uma maneira eficiente de garantir a consistência lógica do banco de dados. Por outro lado, neste método consultas não sofrem nenhuma restrição, da mesma forma que nos projetos tradicionais de banco de dados [TF]. Convém observar neste ponto que, embora o método esteja baseado no modelo relacional, esta não é uma opção essencial.

Projetar banco de dados modularmente não é uma estratégia nova, mas todas as referências conhecidas [DMW,EKW,LMWW,SFNC,SNF,We] tendem a explorar os princípios e a teoria de modularização. A contribuição maior deste trabalho está, portanto, na transformação desta estratégia em um método prático de projeto.

PROJETO MODULAR DE BANCOS DE DADOS

Módulos e Construtores

Um esquema de relação é uma sentença da forma $R[A_1,\ldots,A_n]$, onde R é o nome do esquema e $A_1,\ldots,A_n$ são os atributos do esquema. Uma restrição de integridade é uma sentença da forma n:Q, onde n é o nome da restrição e Q é uma fórmula sobre os esquemas de relação em questão. Uma operação é um procedimento definido em alguma linguagem de programação adequada. Usaremos a notação $f(x_1,\ldots,x_n)$: s para indicar uma operação com n argumentos cujo nome é f.

Um módulo é uma tripla M = (ER,RI,OP) onde

1. ER é um conjunto de esquemas de relação cujos nomes não coincidem.

2. RI é um conjunto de restrições de integridade sobre ER.
   RI deverá conter, para cada esquema de relação $R[A_1,\ldots,A_n]$, um axioma de esquema de relação da forma

   $$\forall x_1 \ldots \forall x_n (R(x_1,\ldots,x_n) \Rightarrow A_1(x_1) \,\&\ldots\&\, A_n(x_n)),$$

   indicando que a interpretação de R deverá ser um subconjunto do produto cartesiano das interpretações de $A_1,\ldots,A_n$.

3. OP é um conjunto de operações sobre LM.

Um módulo pode ser tanto primitivo, se for definido diretamente sem referência a outros módulos, quanto derivado, se for definido com o auxílio de outros módulos previamente definidos através de dois construtores de módulos, subjugamento e extensão.

Um módulo primitivo M=(ER,RI,OP) é definido por uma declaração da forma:

(1) módulo M
esquemas ER;
restrições RI';
operações OP;
responsabilidades RE;
fim-do-módulo

onde RI' é RI sem os axiomas de esquemas de relação (como tais restrições de integridade são completamente fixadas por ER, podem ser omitidas de RI') e RE é um conjunto de cláusulas de responsabilidade da forma 'O garante I' onde O é o nome de uma operação e I é o nome de uma restrição de integridade de M.

O projetista do banco de dados deverá incluir uma cláusula de responsabilidade 'O garante I' sempre que a definição da operação O levar em consideração a restrição de integridade I, ou seja, sempre que alguma mudança na definição de I afetar a definição de O. Esta informação adicional explicita um relacionamento entre operações e restrições que seria extremamente difícil de se obter por uma análise mecânica da definição das restrições e operações.

No resto desta seção, sejam $M_i = (ER_i, RI_i, OP_i)$, i=1,...,n, módulos.

Considere agora o construtor de subjugamento. Intuitivamente, se o projetista define um módulo M como subjugando os módulos $M_1,\ldots,M_n$, então M poderá conter novos esquemas de relação, novas restrições de integridade e novas operações. O módulo M sempre herdará todos os esquemas de relação e restrições de integridade de $M_1,\ldots M_n$. Além disto, M também herdará todas as operações de $M_1,\ldots,M_n$, exceto aquelas que violarem alguma das novas restrições introduzidas em M. Tais operações não mais serão visíveis aos usuários, ou seja, não mais poderão ser chamadas diretamente pelos usuários. Os módulos $M_1,\ldots,M_n$ se tornarão então inacessíveis aos usuários e não mais poderão ser usados para definir outros módulos.

A seguinte declaração define um novo módulo M por subjugamento sobre $M_1,\ldots,M_n$:

(2) módulo M subjuga $M_1,\ldots,M_n$ com
esquemas ER_0;
restrições RI_0;
operações OP_0;
responsabilidades RE;
encobrimentos EN;
fim-de-módulo

onde:

1. ER_0 é um conjunto de esquemas de relação tal que nenhum nome em ER_0 ocorre em $M_1,\ldots,M_n$, e nenhum dos esquemas em ER_0 tem o mesmo nome;
2. RI_0 é um conjunto de restrições de integridade sobre $ER_0, ER_1, \ldots ER_n$;
3. OP_0 é um conjunto de operações podendo chamar como subrotinas apenas as operações no conjunto OP, onde OP é a união de $OP_1,\ldots,OP_n$;
4. EN é um conjunto, possivelmente vazio, de cláusulas de encobrimento da forma 'O pode violar $C_1,\ldots C_k$' onde O é o nome de uma operação de MI, para algum i em [1,n], e C_j é o nome de uma restrição definida em RI_0, para cada j em [1,k]. Dizemos neste caso que O foi encoberta em M.

5. RE é um conjunto de cláusulas de responsabilidade da forma 'O garante I' onde O é o nome de uma operação definida em M e I é um nome de uma restrição também definida em M.

Diz-se que cada objeto de M_i é importado por M.

Mais precisamente, a declaração em (2) define um módulo M=(ER,RI,OP) onde

1. ER é a união de $ER_0,\ldots,ER_n$ e RI é a união de $RI_0,\ldots RI_n$
2. OP é a união de $OP_0, OP'_1,\ldots,OP'_n$, onde $OP'_1,\ldots,OP'_n$ sem as operações encobertas em M, para cada i=1,...,n

Passemos agora à definição do construtor de extensão. Informalmente, um módulo M estende os módulos $M_1,\ldots,M_n$ se cada esquema de relação de M for uma visão sobre os esquemas de relação de $M_1,\ldots M_n$ e cada restrição de integridade de M for uma conseqüência lógica das restrições de $M_1,\ldots,M_n$. Além disto, para se evitar o chamado problema de atualização em visões [FC], a definição de M contém, para cada operação p em uma visão, uma implementação de p em termos das operações de $M_1,\ldots M_n$. Ao contrário do subjugamento, os módulos $M_1,\ldots M_n$ permanecem acessíveis após a definição de M.

Um novo módulo M é definido por extensão sobre $M_1,\ldots,M_n$ através de uma declaração da forma:

(3) módulo M_0 estende $M_1,\ldots,M_n$ com
esquemas ER_0;
restrições RI_0;
operações OP_0;
usando
visões VI;
sub-rogados SR;
fim-do-módulo

onde:

1. a tripla (ER_0,RI_0,OP_0) define o módulo M.
2. VI contém, para cada esquema $R[A_1,\ldots A_k]$ em ER_0, uma definição de visão da forma $R[A_1,\ldots,A_k]$: Q, onde Q é uma fórmula sobre $ER_1,\ldots ER_n$ com k variáveis livres, ordenadas $x_1,\ldots,x_k$. A fórmula $\forall x_1\ldots\forall x_k(R(x_1,\ldots,x_k) <=> Q)$ é chamada de um axioma de definição de visão.
3. SR contém, para cada operação $f(y_1,\ldots,y_m)$: r em OP_0, um sub-rogado, que é uma operação da forma $f(y_1,\ldots,y_m)$: s chamando apenas operações do conjunto $OP = OP_1 U\ldots U OP_n$.

A declaração em (3) define então um novo módulo $M=(ER_0,RI_0,OP_0)$ e acopla M a $M_1,\ldots,M_n$ através do par (VI,SR). Uma sentença $R[A_1,\ldots,A_k]$: Q em VI indica que Q define R em termos dos esquemas de relação de $M_1,\ldots,M_n$. Logo, uma consulta sobre R será traduzida em uma consulta sobre os esquemas de relação de $M_1,\ldots,M_n$ através de Q. Da mesma forma, um sub-rogado $f(y_1,\ldots,y_m)$: s em SR descreve uma implementação de $f(y_1,\ldots y_m)$: r em termos das operações de $M_1,\ldots M_n$. Assim, uma chamada para f gera uma execução de s, e não de r.

Um esquema conceitual modular consiste de um conjunto de módulos que deve satisfazer a uma série de regras de projeto garantindo que, se o banco for atualizado apenas pelas operações visíveis aos usuários, o estado do banco sempre se manterá consistente. Mais precisamente, o conjunto dos esquemas conceituais modulares consistentes e o conjunto dos módulos ativos de cada esquema conceitual são definidos recursivamente como:

1. O conjunto vazio é um esquema conceitual modular consistente com um conjunto vazio de módulos ativos;

2. Seja D um esquema conceitual modular consistente e A o conjunto dos módulos ativos de D. Seja M um módulo tal que nenhum módulo em D tenha o mesmo nome que M. Então D' = D U {M} é um esquema conceitual modular consistente se e somente se M satisfizer a uma das condições abaixo:

   a. se M for um módulo primitivo, então M deverá satisfazer o Requisito 1 (a lista completa dos requisitos encontra-se na Figura 2.1 no final desta seção). O conjunto dos módulos ativos de D' será A' = A U {M}

   b. se M for um módulo definido por extensão sobre $M_1,...M_n$, então M deverá satisfazer os requisitos 2,3,4,5. O conjunto dos módulos ativos de D' será A' = A U {M}

   c. Se M for um módulo definido por subjugamento sobre $M_1,...M_n$, então:

      1) os nomes dos novos esquemas de relação definidos em M deverão ser diferentes dos nomes dos esquemas de relação de $M_1,...M_n$.

      2) M deverá satisfazer os requisitos 6,7,8,9.

      3) O conjunto dos módulos ativos de D' será A' = A U {M} - $\{M_1,...,M_n\}$

Sempre que não qualificarmos um esquema conceitual D, assumiremos que D é consistente.

Seja D um esquema conceitual modular e A o conjunto de módulos ativos de D. O conjunto C de módulos conceituais de D é o subconjunto de A consistindo de todos os módulos primitivos e de todos os módulos ativos definidos por subjugamento; o conjunto E dos módulos externos de D é o conjunto de todos os módulos definidos por extensão em D. Uma operação p de D é ativa, conceitual ou externa se e somente se p for uma operação de um módulo ativo, conceitual ou externo de D, respectivamente.

Um usuário tem em princípio acesso a todos os módulos ativos de um esquema conceitual modular. Logo, verá todos os esquemas de relação e restrições de integridade definidos em todos os módulos, mas poderá atualizar o banco apenas através das operações ativas. Além disto, poderá consultar livremente qualquer esquema de relação. O ABD naturalmente poderá restringir os privilégios de um usuário permitindo o seu acesso a apenas um subconjunto dos módulos ativos, talvez mesmo a apenas alguns módulos definidos por extensão.

Quanto ao processo em si de projetar esquemas conceituais modulares, o método sugerido segue a definição formal. O projetista gradualmente adicionará novos módulos a um esquema modular inicialmente vazio, preocupando-se em como satisfazer as regras de projeto (veja [TFC]).

Para concluir esta seção, enunciamos um teorema que afirma ser a escolha dos requisitos suficiente para garantir preservação de consistência.

TEOREMA 2.1 [TCF]: Seja D um esquema conceitual modular. Suponha que D seja consistente, ou seja, que satisfaça aos requisitos 1 a 9. Então, toda operação ativa de D preserva consistência com respeito a todas as restrições de integridade definidas em módulos de D.

FIGURA 2.1: LISTA DE REQUISITOS

REQUISITO PARA MÓDULOS PRIMITIVOS

Requisito 1: cada operação definida em M não poderá violar nenhuma das restrições de integridade definidas em M.

REQUISITOS PARA MÓDULOS DEFINIDOS POR EXTENSÃO

Seja M um módulo definido por extensão sobre os módulos $M_i=(ER_i,RI_i,OP_i)$, i=1,...,n. Sejam ER_0,RI_0,OP_0,VI e SR os novos esquemas de relação, restrições de integridade, operações, definições de visões e sub-rogados, respectivamente, definidos em M.

Requisito 2: se $f(y_1,...,y_m)$: s é o sub-rogado de $f(y_1,...y_m)$: r definido em SR então s deve ser uma tradução fiel de r [FC].

Requisito 3: se $f(y_1,...,y_m)$: s é um sub-rogado definido em SR, então s só poderá modificar os valores dos esquemas de relação de $M_1,...,M_n$ através de chamadas para as operações definidas em $M_1,...,M_n$.

Requisito 4: para cada restrição de integridade I em RI_0, I' deve ser uma conseqüência lógica das restrições de integridade de $M_1,...,M_n$, onde I' é obtido de I substituindo-se cada fórmula atômica da forma $R(t_1,...,t_k)$ por $Q[t_1/x_1,...,t_k/x_k]$, onde $R[A_1,...,A_k]$: Q é a definição da visão R descrita em VI, e a lista de variáveis livres de Q é $x_1,...,x_k$.

Requisito 5: $M_1,...,M_n$ devem ser módulos ativos de D.

REQUISITOS PARA MÓDULOS DEFINIDOS POR SUBJUGAMENTO

Seja M um módulo definido por subjugamento sobre os módulos $M_i=(ER_i,RI_i,OP_i)$, i=1,...,n. Sejam ER_0,RI_0,OP_0,EN os novos esquemas de relação, restrições de integridade, operações e encobrimentos, respectivamente, definidos em M. Seja RI a união de $RI_0,...,RI_n$ e OP a união de $OP_0,OP_1',...,OP_n'$, onde OP_i' é o conjunto OP_i, excetuando-se aquelas operações que foram encobertas em M, para i=1,...,n.

Requisito 6: cada operação em OP preserva consistência com respeito às restrições de integridade em RI_0.

Requisito 7: cada operação em OP_0 só pode modificar os valores de esquemas de relação em $M_1,...,M_n$ através de chamadas para as operações definidas em $M_1,...,M_n$.

Requisito 8: D não pode conter um módulo definido por extensão usando M_i, para algum i em [1,n].

Requisito 9: $M_1,...,M_n$ devem ser módulos conceituais de D (ou seja, módulos ativos de D que não foram definidos por extensão).

DEFINIÇÃO DO DICIONÁRIO

Nesta seção descreveremos um dicionário contendo os objetos - módulos, esquemas, restrições e operações - definidos em um esquema conceitual modular bem como os relacionamentos entre estes objetos. A descrição do dicionário captura vários conceitos introduzidos na Seção 2 e provê a base para a ferramenta de projeto descrita na Seção 4.

O esquema conceitual do dicionário será descrito em termos do modelo de entidades e relacionamentos. Embora não seja essencial, consideraremos que o dicionário contém apenas as entidades e relacionamentos derivadas de um único esquema conceitual modular D. É importante observar ainda que todas as entidades e relacionamentos e seus atributos podem ser identificados observando apenas a sintaxe dos módulos que compõem D.

O esquema conceitual é o seguinte (usaremos $B(A_1,\ldots,A_n)$ para indicar tanto um conjunto de entidades com nome B e conjunto de atributos $A_1,\ldots,A_n$, quanto para indicar um relacionamento, sem atributos e cujo nome é B, sobre os conjuntos de entidades cujos nomes são $A_1,\ldots,A_n$):

CONJUNTOS DE ENTIDADES

é-primitivo(nome), é-sub(nome) e é-externo(nome)
a cada módulo primitivo, definido por subjugamento ou definido por extensão do esquema conceitual modular em questão, D, corresponde uma entidade no conjunto é-primitivo, é-sub ou é-externo, respectivamente. O único atributo é o nome do módulo.

módulo(nome)
generalização dos três conjuntos anteriores. O único atributo é o nome do módulo.

esquema(nome, lista,def)
a cada esquema de relação definido em um módulo de D corresponde uma entidade neste conjunto. Os atributos são o nome e a lista de atributos do esquema de relação, bem como o mapeamento que o define, se o esquema pertencer a um módulo definido por extensão, ou um valor indefinido, nil, em caso contrário.

restrição(nome,def)
a cada restrição de integridade definida em um módulo de D corresponde uma entidade neste conjunto. Os atributos são o nome e a fórmula que a define.

operação(nome,def,sub-rogado)
a cada operação definida em um módulo de D corresponde uma entidade neste conjunto. Os atributos são o nome e a rotina que define a operação, bem como o sub-rogado que a define, se a operação pertencer a um módulo definido por extensão, ou um valor indefinido, nil, em caso contrário.

RELACIONAMENTOS

subjuga(módulo,módulo) e estende(módulo,módulo)
o par (M,M') estará no relacionamento subjuga ou estende se e somente se M e M' representam dois módulos tais que M é definido por subjugamento ou por extensão, respectivamente, sobre M'.

é-esquema-definido-em(esquema,módulo)
o par (A,M) estará no relacionamento é-esquema-definido-em se e somente se A representa um esquema definido no módulo representado por M.

é-restrição.definida-em(restrição,módulo)
(mesmo, quando A é restrição definida em M).

é-operação-definida-em(operação,módulo)
(mesmo, quando A é operação definida em M). O par (O,M) estará no relacionamento.

é-visão-sobre(esquema, esquema)
o par (V,S) estará no relacionamento é-visão-sobre se e somente se V representa uma visão cujo mapeamento de definição menciona o esquema representado por S.

é-restrição-sobre(restrição,esquema)
o par (I,S) estará no relacionamento é-restrição-sobre sobre se e somente se I representa uma restrição cuja definição menciona o esquema representado por S.

é-operação-sobre(operação,esquema)
o par (O,S) estará no relacionamento é-operação-sobre se e somente se O representa uma operação cuja definição ou cujo sub-rogado (se O for uma operação definida em um módulo introduzido por extensão) mencionam o esquema representado por S.

garante(operação,restrição)
o par (O,I) estará no relacionamento garante se e somente se a definição da operação O levar em consideração a restrição representada por I.

pode-violar(operação,restrição)
O par (O,I) estará no relacionamento pode-violar se e somente se O representa uma operação cuja execução poderá violar a restrição I.

chama(operação,operação)
o par (O,O') estará no relacionamento chama se e somente se O representa uma operação cuja definição ou cujo sub-rogado (se O for uma operação definida em um módulo introduzido por extensão) mencionam a operação representada por O'.

UM SISTEMA ESPECIALISTA PARA PROJETO DE BANCO DE DADOS

Nesta seção descreveremos resumidamente o protótipo de uma ferramenta de software que auxilia o Administrador de Banco de Dados (ABD) adicionar, interativamente, módulos a um esquema de banco de dados. Este protótipo implementa parcialmente o dicionário descrito na Seção 3.

O protótipo é um exemplo de um assistente especialista cujo conceito foi apresentado em [FM] para designar ferramentas inteligentes e relativamente pequenas que auxiliam no projeto, uso e manutenção de grandes sistemas convencionais. A versão corrente desta ferramenta roda em computador da família do IBM PC e foi escrita usando o apes [HS] que é uma extensão do micro-PROLOG [CM].

Representaremos parte do dicionário através de uma série de axiomas, que serão literais positivos sobre um único símbolo predicativo binário, 'tab'. Cada axioma corresponde ou a um relacionamento ou a uma entidade designando módulos de um esquema de banco de dados. Um axioma correspondente a um relacionamento consiste, intuitivamente, de duas listas: a primeira denota o tipo de relacionamento, expresso pelos tipos de entidades envolvidas, e a segunda indica o relacionamento em si, expresso pelos nomes das entidades. O formato geral deste tipo de axioma é (usando notação infixa para 'tab'):

(<tipo> <tipo>) tab (<nome> <nome> <versão>)

onde <versão> é um inteiro positivo (sempre 1, quando o dicionário é criado).

Um axioma representando um módulo tem o seguinte formato:

(mod) tab (<nome> <tipo> <versão>)

onde <tipo> pertence ao conjunto primitivo, subjugamento, extensão).

Na tabela 4.1 apresentamos a correspondência entre as entradas do dicionário e suas representações como axiomas, conforme foi implementado pela ferramenta.

Tabela 4.1 - Representação do Dicionário

Entrada no Dicionário	Formato do Axioma
é-primitivo(M)	(mod) tab (M 'primitivo' n)
é-sub(M)	(mod) tab (M 'subjugamento' n)
é-externo(M)	(mod) tab (M 'extensão' n)
esquema(S,L,Q)	não implementado
restrição(I,Q)	não implementado
operação(O,P,P')	não implementado
M subjuga M'	(mod mod) tab (M M' n)
M estende M'	(mod mod) tab (M M' n)
S é-esquema-definido-em M	(sch mod) tab (S M n)
I é-restrição-definida-em M	(con mod) tab (I M n)
O é-operação-definida-em M	(ope mod) tab (O M n)
V é-visão-sobre S	(sch sch) tab (V S n)
I é-restrição-sobre S	(con sch) tab (I S n)
O é-operação-sobre S	(ope sch) tab (O S n)
O garante I	(ope con) tab (O I n)
O pode-violar I	(ope con) tab (hid O) I n)
O chama O'	(ope ope) tab (O O' n)

Nota: n é o número da versão

Graças ao uso de apes, o protótipo é altamente interativo. Esboçaremos a seguir como ele pode ser usado pelo ABD para adicionar um novo módulo a um esquema de banco de dados.

Para iniciar a definição de um módulo, o ABD tecla module <nome>. A partir deste ponto, o protótipo induz o ABD a fornecer todas as informações necessárias à definição dos esquemas, das restrições e das operações do módulo. O "programa" consiste de um predicado chamado "module" que por sua vez chama outros predicados para criarem os vários componentes do módulo. Um módulo particular pode ou não ter esquemas, restrições ou operações. Entretanto:

- se o módulo M não é primitivo, o ABD deve listar os módulos que M subjuga ou estende;
- se o módulo M é definido por extensão, cada esquema S é uma visão. Assim, o ABD deve definir o mapeamento de S sobre os esquemas dos módulos estendidos por M;
- para cada restrição ou operação O, o ABD deve listar todos os esquemas que O referencia;
- somente operações de módulos não-primitivos podem chamar outras operações; além disso, todas as operações dos módulos criados por extensão são sub-rogados e devem, portanto, incluir tais chamadas. O ABD deve então informar os relacionamentos do tipo chama.

Assim, a presença de certos relacionamentos (indicados pela adição das entradas correspondentes em tab) é compulsório, e o predicado "module" falhará se o ABD declarar que eles não existem (teclando "end" em resposta a uma pergunta apresentada pelo protótipo).

O protótipo determina, de forma procedural, a seqüência a ser seguida pelo ABD na criação dos vários relacionamentos e a natureza compulsória ou opcional destes. Por outro lado, usando as facilidades de 'unique-answer' e 'valid-answer' do apes, o protótipo define separadamente, de forma declarativa, que critérios decidem se os valores fornecidos pelo ABD, como respostas, são aceitáveis.

Apresentamos abaixo, por tipo de relacionamento criado, os critérios que são atualmente garantidos.

(mod) tab (x y 1)	y ∈ {primitivo, subjugamento, extensão} 584
(mod mod) tab (x y 1)	y é um módulo ativo, que não deve ter sido criado por extensão nem estendido se x está sendo criado por subjugamento
(sch sch) tab (x y 1)	o esquema y é acessível para algum módulo usado na definição do módulo no qual a visão x está sendo definida
(con sch) tab (x y 1)	o esquema y é acessível para algum módulo no qual a restrição x está sendo definida
(ope ope) tab (x y 1)	a operação y é acessível para algum módulo usado na definição do módulo no qual a operação x está sendo definida; se esta última é definida por extensão, y é relacionado a algum esquema que suporta sua visão
(ope sch) tab (x y 1)	o esquema y é acessível para o módulo onde a operação x está sendo definida
(ope con) tab (x y 1)	a operação x e a restrição y tem algum esquema em comum
O mesmo, para x= (hid z)	a operação z é chamada por uma operação dá qual a restrição y depende.

O protótipo formula as perguntas relevantes ao ABD através de sentenças em linguagem natural, e adota menus estáticos e dinâmicos para restringir as suas respostas. O protótipo assegura que os nomes no dicionário são únicos. Para estes propósitos são usadas algumas facilidades adicionais do apes ('which-template', 'is-menu', 'is-template').

Se olharmos para a Figura 2.1 no final da Seção 2, podemos comparar os critérios implementados com os requisitos necessários para um projeto modular correto. Os Requisitos 1, 2, 4, 6 e 7 não são garantidos por requererem uma descrição detalhada dos componentes. Os Requisitos 5, 8 e 9 são explicitamente garantidos pelos critérios implementados. O Requisito 3, que se refere a módulos criados por extensão, é garantido através de restrições sobre operações e visões do módulo em relação a operações e esquemas dos módulos que foram estendidos.

De uma forma geral, podemos certamente fazer mais em termos de verificações na consistência do projeto modular usando as informações que são extraídas do ABD. Entretanto, o que verificamos agora é suficiente para demonstrar a utilidade deste tipo de assistente especialista.

CONCLUSÕES E DIREÇÕES FUTURAS DE PESQUISAS

Descrevemos neste trabalho uma ferramenta de software orientada para o método de projeto modular de banco de dados apresentada primeiramente em [TCF]. O método em si foi aumentado pela incorporação das cláusulas de encobrimento e responsabilidade, e pelo refinamento de algumas regras de projeto.

A ferramenta de software está implementada até o ponto de auxiliar o administrador de banco de dados a adicionar novos módulos a um esquema conceitual existente. O processo de reprojeto, embora não implementado, está especificado em detalhe em [TFC].

Os planos futuros incluem transformar a ferramenta em um sistema de dicionário completo, incorporando o máximo possível de conhecimento sobre a metodologia de projeto.

[CM] K.L. Clark e F.G. McCabe. "micro-PROLOG: programming in logic". Prentice-Hall (1984)

[DMW] W. Dosch, G. Mascari, M. Wirsing. "On the Algebraic Specification of Databases". Proc. 8th Int'L Conf. on Very Large Data Bases (1982)

[EKW] H. Ehrig, H.-J. Kreowski, H. Weber. "Algebraic Specification Schemes for Data Base Systems". Proc. 4th Int'L Conf. on Very Large Data Bases (1978)

[FC] A.L. Furtado e M.A. Casanova. "Updating Relational Views", in "Query Processing in Database Systems", Springer Verlag (no prelo)

[FM] A.L. Furtado and C.M.O. Moura. "Expert helpers to data-based information systems". Proc. of the First International Workshop on Expert Database Systems (1984), 298-313

[HS] P. Hammond and M. Sergot. "apes: augmented PROLOG for expert systems - reference manual". Logic Based Systems Ltd. (1984)

[LMWW] P.C. Lockemann, H.C. Mayr, W.H. Weil, W.H. Wohlleber. "Data Abstractions for Data Base Systems". ACM Transactions on Database Systems 4:1 (1979)

[LZ] B. Liskov, S. Zilles. "Specification Techniques for Data Abstractions". IEEE Transactions on Software Engineering SE-1 (1975)

[Pa] D. Parnas. "On the Criteria to be Used in Decomposing Systems into Modules". Comm. of the ACM 15:12 (1972)

[SFNC] U. Schiel, A.L. Furtado, E.J. Neuhold, M.A. Casanova. "Towards Multi-level and Modular Conceptual Schema Specifications". Inform. Systems 9:1 (1984), 43-57.

[SNF] C.S. dos Santos, E.J. Neuhold, A.L. Furtado. "A Data Type Approach to the Entity-Relationship Model". Int'L. Conf. of the Entity-Relationship Approach to Systems Analysis and Design (1980)

[TCF] L. Tucherman, M.A. Casanova e A.L. Furtado, "A Pragmatic Approach to Modular Database Design", Proc. of the 9th Int'L Conf. on Very Large Data Bases, Florence, Italy (1983), 219-231.

[TFC] L. Tucherman, A.L. Furtado e M.A. Casanova, "A Tool for Modular Database Design", Relatório Técnico do CCB (Centro Científico de Brasília) a ser publicado.

[TF] T.J. Teorey, J.P. Fry. "Design of Database Structures", Prentice-Hall, Inc. (1982)

[We] H. Weber. "Modularity in Data Base Systems Design". Proc. Joint IBM/Univ. Newcastle upon Tyne Seminar (1979)

[ZLT] S.N. Zilles, P. Lucas, J.W. Thatcher. "A Look at Algebraic Specifications". Research Rep. RJ3568 IBM Thomas J. Watson Research Center (1982)

O CONCEITO DE TEMPO EM SGBDs GENERALIZADOS

J. PALAZZO M. DE OLIVEIRA*

SUMARIO

O conceito de tempo é necessário para o desenvolvimento de aplicações administrativas, de economia e particularmente de automação de escritórios. Apesar de sua importância atualmente os SGBD não oferecem explicitamente este tipo de dado. Um modelo para a representação do tempo é apresentado bem como exemplos da linguagem de manipulação associada. Estes conceitos estão sendo utilizados para o desenvolvimento de sistema como a agenda eletrônica e o controle de projetos. Mesmo antes de sua disponibilidade em produtos comerciais estes conceitos são de grande valor no desenvolvimento de sistemas de aplicação.

ABSTRACT

The concept of time is needed in the development of computerized systems in economics, manegement systems and particularly for office automation system. Besides their significance, DBMS today do not explicity deel with this type of data. A model for representing the time is describe as well as examples of the associated manipulation language. These tecniques are being used for the development of software systems such as project management and automatic diary. Even before supplied marketable products these concepts are of great value for the development of application Systems.

* Doutor em Informática pela Universidade de Grenoble, França; Mestre em Ciência da Computação pela UFRGS e Engenheiro Eletricista pela UFRGS. Áreas de interesse Bancos de Dados, Sistemas de Apoio a Decisão, Sistemas Distribuidos e Automação de Escritórios CPGCC/UFRGS. Caixa Postal 1501, 90000 Porto Alegre, Brasil.

INTRODUÇÃO

A necessidade de manipular o tempo foi evidenciada há muitos anos por aplicações diversas. Nos bancos de dados empregados para aplicações administrativas o tempo intervem no processo de modelização para representar uma certa realidade: datas de vencimentos de títulos, períodos de férias, prazos de carência de empréstimos etc. Nas aplicações econômicas ele permite representar modelos econométricos incluindo séries históricas.

Em particular na Automação de Escritórios (AE) a gestão do tempo é uma necessidade importante. Uma aplicação de AE trata particularmente da gestão de textos, das comunicações e do tempo. A gestão de documentos foi tratada em /PAL 84b/ e deverá ser abordada em maior profundidade em um próximo artigo. O conceito de tempo é um fator importante para a modelização das atividades de escritório e para o controle de projetos.

Neste artigo é tratado o conceito de tempo de proposto um modelo para sua definição e manipulação.

O TEMPO E OS BANCOS DE DADOS

Nos sistemas tradicionais o conteúdo da base representa um instântaneo do estado do ambiente de uma aplicação |HAM 81|. Os SGBD atuais oferecem ao usuário a versão mais recente dos objetos representados na base. A cada atualização de um atributo o valor anterior é destruído. A representação de valores sucessivos pode ser feita de uma forma manual através de arrays ou campos repetitivos de registros.

Por outro lado as transações são estreitamento ligadas ao conceito do tempo. No caso da AE as transações apresentam características especiais pois ão definidas sobre grandes objetos complexos - os documentos - e desenrolam-se durante períodos consideráveis de tempo |LOR 82|. Por exemplo: a redação de uma secção de um manual pode extender-se por dias e durante este período nenhuma outra alteração deverá ser permitida sobre esta secção. A existência do conceito de tempo explícito no modelo de dados é um auxílio para a gestão deste tipo de transações.

Nos sistemas tradicionais o tempo é representado por um inteiro como o número de segundos a partir de um certo instante (clock). Se esta solução é bem adaptada aos problemas físicos ela não o é aos problemas administrativos. Nesta classe de aplicações é empregado um calendário. Um calendário é uma associação de períodos, que podem ser hierárquicos como mês, dia, hora, à representação interna do tempo.

Deve-se notar que os pontos referenciados por um calendário (datas) são na verdade períodos |FAL 74|. Com a intrdução do conceito de calendário se apresentam problemas para a manipulação dos intervalos pois as operações podem ser definidas sobre intervalos

de diferentes granularidades.

A partir da definição do tipo tempo é possível indentificar duas classes de atributos os estáticos e os dinâmicos. Um atributo estático é invariável em relação ao tempo, por exemplo a data de nascimento de uma pessoa. Os atributos dinâmicos são variáveis como o endereço que muda diversas vezes na vida de uma pessoa. É possível definir, também, a manutenção de diferentes versões de um objeto da base sendo que cada versão está associada a um instante do passado. Esta geração de diferentes versões de um mesmo objeto é conhecida como instantâneos (snapshots) |ADI 80|.

Um banco de dados que mantém os estados sucessivos de certos atributos ou objetos é denominado Banco de Dados Histórico (Historical Database). Um banco de dados histórico permite consultas sobre instantes definidos de sua existência. Um exemplo é a consulta de uma versão anterior de um documento.

Diversos estudos para a formalização do conceito de tempo foram desenvolvidos e são referenciados e os mais importantes comentados em |BAD 82|.

CLASSIFICAÇÃO DAS METODOLOGIAS

As diferentes possibilidades para a implementação de extensões temporais em bancos de dados podem ser classificados em categorias. O critério adotado é a integração crecente do conceito de tempo ao sistema. Na primeira categoria a manipulação do tempo é externa ao SGBD. Na segunda o conceito de tempo é associado ao SGBD por rotinas específicas e finalmente na última o tempo pertence ao modelo de dados. Estas categorias são descritas a seguir:

1. a manipulação do tempo é feita de forma explícita pelos programas de aplicação. O SGBD, ou mais geralmente o sistema de arquivos, só pode armazenar dados de tipo tradicional como inteiros, reais ou cadeias de caracteres. Toda a semântica associada ao tempo pertence à estrutura lógica dos programas. Neste nível o usuário deve conhecer a semântica associada ao tempo e assegurar a validade das operações sobre os dados definidos sobre este tipo. A integridade da base é também responsabilidade do usuário. Um sistema desta categoria é apresentado em |WIE 75| Time Oriented Database, desenvolvido na Universidade de Stanford.

2. diferentes extensões para a manipulação de dados com uma semântica extendida são realizadas por programas chamados "experts". Esta solução apresenta a vantagem de uma modificação mínima do SGBD subjacente. M. Stonebraker |OVE 82| faz referência a uma modificação de somente 62 linhas de código fonte para permitir ao sistema INGRES a gestão dos programas "experts".

Para o usuário a semântica de um tipo de dados extendido é manipulada automaticamente. A maior fraqueza deste tipo de solução é a heterogeneidade na representação dos tipos extendidos. Para sua manipulação é necessária a chamada explícita do expert associado e por outro lado é impossível a representaçao do tempo no esquema conceitual da base.

3. finalmente é possível escolher um modelo de dados e definir uma extensão para a gestão do tempo |KLO 83|. Nesta solução a semântica se torna estrutural, ela pertence ao modelo de dados. A definição de um esquema contém a representação explícita do tempo e dos conceitos associados.

Na definição de um sistema de Banco de Dados Generalizado |PAL 84b| para aplicações administrativas e em particular para aplicações de automação de escritórios a extensão para a manipulação do tempo é uma das que apresentam maior interesse.

O MODELO PARA A REPRESENTAÇÃO DO TEMPO

A seguir é apresentada a extensão realizada sobre o modelo de dados TIGRE |PAL 84a| , |VAE 84| para a representação do tempo. Esta descrição evidencia os diversos aspectos desta representação e apresenta as operações associadas.

O modelo de base é o Entidade-Relacionamento |CHE 76| sendo que a descrição dos tipos de dados e das entidades e relacionamentos é feita de forma análoga a definição de tipos e variáveis em Pascal.

São definidos diversos tipos de base: o tempo simples, os tipos restritos, os intervalos, as durações, os periódicos e as constantes. Para cada tipo serão apresentados exemplos. A sintaxe completa de cada tipo bem como dos enunciados de manipulação pode ser ser encontrada em |ABP 84|.

O TIPO DE TEMPO SIMPLES

Ele será designado pelo termo time e definido por meio dos períodos do calendário Gregoriano aos quais são adicionados a hora, o minuto e o segundo. Um valor deste tipo é, por exemplo: 1985/01/12 18h14:37. Foi escolhida como representação externa de time uma cadeia de caracteres de comprimento 19. Na definição deste tipo é possível referenciar o instante presente graças à palavra chave present-time. Esta referência produz a execução de uma rotina que lê o relógio do computador.

OS TIPOS RESTRITOS

O nível de representação mais fino, ou seja a granularidade, é constituído pelo segundo. Em muitas aplicações não é necessária uma granularidade tão fina. Nestes casos é possível definir tipos restritos:

tyte data : time > hour

esta definição especifica um calendário do tipo (ano, mês, dia) e são possíveis declarações do tipo:

data-de-compra : data

que pode assumir o valor '1985/01/14'

No caso dos tipos restritos os operadores de comparação são definidos somente sobre operandos de mesma precisão para evitar a aparição de valores indeterminados como resultado de uma comparação.

TIPO INTERVALO

Um intervalo de tempo é definido como um período entre datas sendo a primeira menor que a segunda e definidas sobre tipos de mesma granularidade. Por definição os extremos são incluídos no intervalo e um deles pode ser present-time.

Exemplo: type férias: time-interval

e pode assumir o valor de "1985/02/01 - 1985/02/28".

TIPO DURAÇÃO

A noção de duração é encontrada correntemente nas aplicações. Por exemplo a duração do mandato do presidente da República é de 6 anos, a duração das férias é de 30 dias e a semana de trabalho 48 horas. Para representar este conceito existe o tipo duração. Por exemplo:

type duração - férias : duration

TIPO PERÍODO

Um exemplo é o período de almoço entre às 12h e às 13h ou a emissão de um extrato bancário todos os dias 28 do mês. Para representar as atividades que se repetem periodicamente e possível definir o tipo periódico. Uma variável definida sobre este tipo pode assumir, por exemplo, os valores: "mês:=12" para definir os meses de dezembro, "dia:=10-dia :=20" para definir o intervalo periódico correspondente aos dias compreendidos entre o dia 10 e o dia 20 de cada mês.

CONSTANTES

Para facilitar a definição do esquema e para manipular mais facilmente dados temporais podemos definir constantes do tipo duração, por exemplo:

time-constant semestre = 6 month,
semana = 7 day;

Consideremos as seguintes questões: (1) quais os salários de um funcionário nos últimos 12 meses? (2) qual o salário do funcionário X em janeiro de 80? (3) qual o histórico completo dos salários do funcionário y?

A questão (1) é relativa, no sentido de que não faz referência explicitamente a um instante do passado, e se refere ao período de 12 meses que termina no instante presente. A questão (2) faz referência explicitamente a um instante preciso do passado enquanto que a (3) é uma extensão da primeira e corresponde desta vez explicitamente as versões sucessivas do salário.

Como foi visto, a partir dos exemplos, a noção de históricos é ligada a aspectos de intervalos e igualmente as modificações dos valores. Neste modelo os históricos tem duas facetas a periodicidade da amostragem e a persistência dos valores na base. A periodicidade corresponde ao número de vezes, ao longo do tempo, que é desejado manter os valores sucessivos. A persistência se relaciona ao número de valores históricos que devem ser armazenados na base. Uma persistência nula corresponde ao caso tradicional onde apenas o último valor é disponível. Uma persistência indefinida corresponde a uma lista teoricamente ilimitada de valores. Consideremos alguns exemplos:

```
type salário: dynamic
              each month
              last 12 real;
```

A palavra chave dynamic indica que se trata de um dado do tipo histórico. A periodicidade é indicada pela cláusula each month e a cláusula last 12 indica que os últimos 12 valores serão mantidos na base. Um outro exemplo é a definição de uma entidade denominada câmbio constituída por dois atributos: a moeda e a taxa:

```
type câmbio entity
     moeda: string (12);
     taxa:  dynamic
            real
     end;
```

esta declaração associa ao atributo taxa uma periodicidade indefinida. O conceito de histórico pode ser, também, aplicado as entidades e associações permitindo, assim, sem o uso da linguagem de manipulação definir instantâneos próximos aos conceitos definidos em |ADI 81|.

Exemplo:

```
type fornecedor: relationship
                 dynamic each year
                 between empresa and artigo
     end;
```

é a definição de um relacionamento para o qual o sistema mantém um instantâneio a cada ano para cada ocorrência do tipo.

A LINGUAGEM DE MANIPULAÇÃO

A extensão do modelo de dados descrita corresponde a extensão da linguagem de manipulação. Nesta seção serão apresentados alguns exemplos:

```
select c.nome, c.endereço
from estudante e
where duração-de-bolsa > 12 month
```

terá como resultado todos os nomes e endereços de estudantes que tenham bolsas com duração maior do que 12 meses.

```
select e.nome
from estudante e of escola ec
where ec.nome-escola = 'UFRGS' and
      e.categoria = 'mestrado' and
      year (e.nasc) = 1959
```

seleciona os nomes de todos os estudantes da UFRGS inscritos em programas de mestrado e que tenham nascido depois de 1959.

```
x: = select e.duração-de-bolsa
     from estudante e
     where e.nome = 'Dupont' and
           e.categoria = 'mestrado';

x: = addtime (x, 5 month);

     replace e.duração-de-bolsa:=x
     from estudante e
     where e.nome 2 'Dupont' and
           e.categoria e 'mestrado'
```

este conjunto de instruções permite a prorrogação de 5 meses à bolsa do estudante de mestrado Dupont.

CONCLUSÕES

Neste artigo foram apresentadas a extensão a um modelo de dados entidade-relacionamento para introduzir explicitamente nos bancos de dados generalizados o conceito de tempo. Este conceito é representado sob seus diversos aspectos: medida do tempo, noção de duração, de intervalo e de históricos.

A disponibilidade destes conceitos permite o desenvolvimento de aplicações de automação de escritórios, tais como agenda eletrônica, de forma integrada ao sistema de banco de dados. Esta integração é um dos fatores de sucesso no desenvolvimento de sistemas de AE e de apoio à decisão.

Como desenvolvimento da pesquisa a implantação das rotinas de manipulação dos conceitos relativos ao tempo está sendo concluída no INPG em Grenoble.

Em paralelo o uso dos conceitos de agregação |PAL 84b| e do tempo servem de base para a definição conceitual e implantação de um sistema integrando a agenda eletrônica e o controle de projetos no CPGCC/UFRGS em Porto Alegre.

|ADI 84| ADIBA, M; BUI, Qn; OLIVEIRA, José Plazzo. Notion de temp dans les bases de données generalisées. Grenoble, IMAG, 1984. (Tigre 23)

|ADI 81| ADIBA, M. Derived relations: a unified mechanism for views, snapshots and distributed data. In; INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 7. , Cannes, Sept. 9-11, 1981. Proceedings. New York, IEEE, 1981. p. 293-305.

|ADI 80| ADIBA, M & LINDSAY, B. Database snapshots. In: INTERNATIONAL CONFERENCE on VERY LARGE DATA BASES, 6., Montreal, Oct. 1.3, 1980. Proceedings. New York, IEEE, 1980. p. 86-91.

|BOU 82| BOULOUR, A. et alii. The role of time in information processing: a surney. SIGMOD Record, 12 (3):27-50 Apr. 82.

|FAL 74| FALKENBERG, Ef. Time handling in data base systems. Stuttgart, Universitat Stuttgart, 1974. (Relatório 7/1974).

|HAM 81| HAMMER, M & MOLEOD, D. Database description with SDM: Semantic Database Model. ACM Trans. on Database Systems, 6(3): 351-86, Sept. 1981.

|KLO 83| KLOPPROGGE, M.R. & LOCKEMANN, P.C. Modelling information preserving databases: consequences of the concert of time. In: INTERNATIONAL CONFERENCE on VERY LARGE DATA BASES, 9., Florence, Oct.31 - Nov. 2, 1983. Proceedings. Florence, VLDB Endowment, 1983. p. 399-416.

|LOR 82| LORIE, R. & PLOFFE, W. Complex objets and their use for the design of transsactions. San Jose, IBM, 1982. (RU 3706/42922).

|OLI 84a| OLIVEIRA, José Palazzo. Um modele de données et sa representation relationelle dans un sistense de gestion de bases de données generalisées. Grenoble, INPG, 1984. (Tese de doutorado)

|OLI 84b| OLIVEIRA, José Palazzo. Banco de dados generalizados: a terceira geração. In: CONGRESSO NACIONAL de INFORMATICA, 17. Rio de Janeiro, nov, 1984. Anais. Rio de Janeiro, SUCESU, 184. v.1. p. 6-14.

|OVE 82| OVERMEYER, R. & STONEBRAKER, M. Implementation of a time expert in a data base system. SIGMOD Record, 12(3):51

|VEL 84| VELEZ, F. Un modele et un language pom les bases de données generalisées. Grenoble, INPG, 1984. (Tese de doutorado).

|WIE 75| WIEDERHOLD, G. et alii. Structured organization of clinical data bases. In: NATIONAL COMPUTER CONFERENCE, Anaheim, May 19-22, 1975. Proceedings. Montreale. AFIPS Press, 1975. p. 479-85.

Un Sistema de Almacenamiento y Recuperación de Información Para Facilitar Tanto Procesos Puntuales Como en Orden Lexicográfico

F. AURTENECHEA *

RESUMEN

Se describe la estructura de hashing extendible para el almacenamiento y recuperación de información en archivos dinámicos. Se presenta una proposición de un esquema de almacenamiento mediante hashing extendible para permitir almacenar y procesar identificadores en orden lexicográfico, sin las restricciones de tiempo incurridas con hashing tradicional.

La factibilidad de este sistema se analiza basicamente con respecto a las necesidades de almacenamiento que la estructura requiere. El problema fundamental consiste en que, en general, no es posible encontrar una función de hashing que distribuya las claves en orden lexicográfico y al mismo tiempo distribuya el espacio de almacenamiento en forma uniforme. Se discute una solución a este problema, que se basa fundamentalmente en particionar el espacio de almacenamiento en bloques de tamaño variable.

ABSTRACT

The extendible hashing structure for information retrieval of dynamic files is described. The paper presents a proposal of a system for structuring data using extendible hashing, to allow keys to be stored and processed in lexicographic order, without the restrictions of traditional hashing.

This system is analyzed, basically, according to the requirements of primary and secondary storage needed for the structure. Tha basic issue is that, in general, it is difficult to find lexicographic order-preserving hash functions that distribute the key space uniformly over an address space. A solution to this problem is presented, which is based in partitioning the address space into variable-length blocks

* Ingeniero Civil Industrial (PUC Chile 1977); Master of Philosophy (M. Phil) University of Edinburgh (1982); Redes de Comunicación de Datos, Ingeniería de Software, Bases de Datos; Departamento de Ciencia de la Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Vicuña Mackenna 4860, casilla 6177, Santiago, Chile.

I Introducción

Los sistemas destinados a almacenar grandes volúmenes de información se pueden evaluar, básicamente, de acuerdo a tres criterios fundamentales: i) el tiempo promedio de respuesta a una determinada consulta, ii) las necesidades de almacenamiento que requieren, y iii) la facilidad de adaptar su estructura en respuesta a demandas externas de inserciones y eliminaciones de registros.

Una estructura muy utilizada como método de almacenamiento y recuperación de información es la estructura de hashing. Cuando se requiere de accesos puntuales a la información, esta estructura es una de las más eficientes con respecto al primero de los criterios anteriormente mencionados (tiempo de respuesta). Sin embargo, esta eficiencia se logra a expensas de sobredimensionar el archivo o tabla de almacenamiento (tabla de hashing). Además, su estructura es básicamente estática, ya que una vez definida la dimensión de la tabla de hashing, sólo se pueden alterar valores de ésta, pero la tabla misma no puede ser modificada. Por lo tanto, si la dimensión de la tabla se subestima con respecto a los requerimientos reales de almacenamiento (por ejemplo, se logra ocupar más del 90% de la tabla), se debe proceder a reconfigurar completamente el archivo que contiene los datos.

En la búsqueda de soluciones para este problema, se han realizado investigaciones en nuevas estructuras de tipo hashing, denominadas hashing extendible o hashing dinámico (Fagin et al,1977 y Larson,1978). Este esquema de hashing, además de ser eficiente en términos de tiempo de respuesta ante consulta de sus elementos (máximo dos accesos al disco), es una estructura dinámica. Esto es, facilita el almacenamiento de la información de acuerdo a las reales necesidades, al permitir que las tablas de hashing puedan expandirse o contraerse dinámicamente.

Hashing extendible es una estructura orientada al almacenamiento de datos en dispositivos de acceso secundario (por ejemplo, discos magnéticos), donde la información se almacena mediante un esquema de páginas de tamaño fijo. El sistema permite mantener relativamente uniforme el factor de ocupación de las páginas, mediante esquemas de reestructuración que involucran un mínimo de costo adicional.

Sin embargo, con este método de almacenamiento, al igual que con el método de hashing tradicional, normalmente no es posible acotar en tiempo real la realización de procesos secuenciales, con respecto al orden lexicográfico de la clave de acceso a la información. En otras palabras, dada una clave cualquiera K_i, el costo de encontrar la clave que precede o sucede a K_i (K_{i-1}, K_{i+1}) en secuencia natural es, en general, difícil de soportar (Knuth,1974, Severance,1976).

II Esquema de Almacenamiento

La estructura de hashing extendible tiene dos propósitos fundamentales: (i) facilitar el almacenamiento de archivos dinámicos permitiendo que las tablas de hashing puedan expandirse o contraerse dinámicamente, y (ii) incorporar un esquema de páginas de almacenamiento, manteniendo uniformidad en la cantidad de datos existente en cada una.

La estructura incorpora dos niveles de almacenamiento. El primer nivel, es un directorio que contiene índices al segundo nivel, siendo este último una estructura de páginas u hojas. El directorio consiste de una tabla de acceso directo, cuyo espacio de direccionamiento D, se obtiene transformando el espacio de claves K mediante una función de hashing h(K) (Carter y Wegman, 1977). Cada entrada en el directorio representa un puntero que direcciona a una página de información, la que agrupa a un cierto número de registros con características similares.

(h(K) ---> D =====> Paginas)

La función de hashing establece una transformación del espacio de claves K en un espacio D de direcciones de gran tamaño. El espacio D se separa en una partición P de m bloques, en que cada bloque tiene una página asignada; el directorio establece la correspondencia entre bloques y páginas. La partición P define m+1 límites (a_0, a_1,..., a_m), en que la página P_i contiene todas las claves K tal que, $a_{i-1} \leq h(K) < a_i$.

Si se llena la página Pi, se puede cambiar la partición, por ejemplo moviendo el límite ai en uno y reorganizando sólo aquellas claves afectadas por esta reestructuración local. Luego, la idea de hacer extendible la tabla de hashing consiste en variar la partición P en un espacio D muy grande, sin variar la función de hashing.

El esquema anteriormente descrito se basa fundamentalmente en el Sistema Buddy para el manejo de información (Knuth 1974).

En este sistema se escoge D = (0,1,...,2^n-1), para algún n fijo. Entonces a_0=0 <a_1 < ... <a_m=2^n-1 son los límites de una partición de un sistema buddy, si y solo si todo intervalo (a_{i-1},a_i) puede ser obtenido por sucesivas divisiones de intervalos en D.

A continuación se describe un esquema específico de hashing extendible (ver Fagin et al,1979), y en el cual se basa nuestro análisis. Se escoge una función de hashing h fija con la cual se obtiene, por cada clave K, una pseudo-clave K'=h(K). El espacio de direccionamiento D, del directorio, es igual a 2^{n-1}, donde n es el tamaño máximo de la pseudo clave, en bits. Así por ejemplo, con 32 bits para definir el espacio D, se pueden direccionar aproximadamente 10^{10} páginas.

Se define como profundidad p del directorio, a los primeros p bits de la pseudo clave K', los que se usan para determinar la página. El directorio tiene 2^p punteros, no necesariamente distintos. El primero indica a la página que almacena todas aquellas claves cuya pseudo-clave comienza con p ceros consecutivos. El segundo puntero identifica a todas las claves cuya pseudo clave es 0...01 en sus primeros p bits; el tercero identifica a aquellas claves cuya pseudo clave es 0...010 en sus primeros p bits, y así sucesivamente en orden lexicográfico. Finalmente, el último puntero identifica a todas las claves cuya pseudo clave comienza con p unos consecutivos. La Fig. 1 muestra un posible esquema de almacenamiento para p = 3.

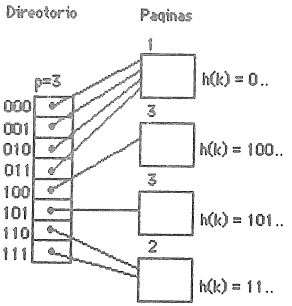
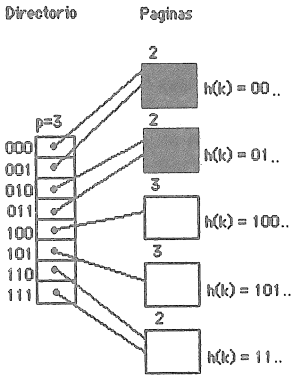


Fig. 1 Un Esquema de Hashing Extendible

Siguiendo el puntero 111 de la Fig. 1, se llega a una página cuya profundidad local (p') es igual a 2. Esto significa que esta página contiene no sólo aquellas claves cuyos primeros 3 bits son 111, sino también aquellas que comienzan con 11. Luego, el puntero 110 también señala a esta página.

Si se llena una página (o por ejemplo, se sobrepasa el 90% de ocupación) en la cual p' < p, ésta se separa en dos páginas, aumentando cada una su profundidad local en 1. Considerando la Fig. 1, tal caso ocurriría si se llena la página cuya pseudo clave comienza con 0.., (su p' es igual a 1). Después de producirse la reconfiguración, como se muestra en la Fig. 2, una página contiene a todas las claves cuya pseudo-clave comienza con 00, y la otra a todas las claves cuya pseudo-clave comienza con 01. Ambas páginas con profundidad local igual a 2.



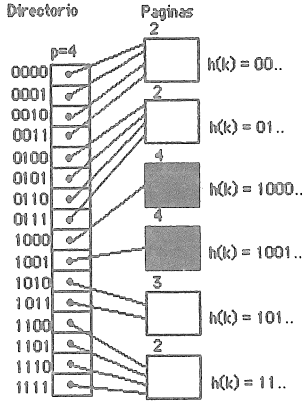


Fig 2 Rebalse de pagina sin duplicar directorio Fig 3. Rebalse de pagina duplicando directorio

Si se llena una página i cuya profundidad local es igual a la profundidad del directorio ($p=p'_1$), entonces éste debe aumentar su profundidad local en 1, lo que significa duplicar su tamaño. Por ejemplo, si se llena la página indicada por el puntero 100, de la Fig. 2, además de separar su información en dos páginas, se debe proceder a duplicar el directorio (ver Fig. 3).

Nótese que a lo más hay dos accesos al disco para encontrar una determinada clave. Uno para localizar la página específica en el directorio, y otro para localizar la página donde se encuentra la clave.

IV Secuencialidad

La distribución del espacio de claves es generalmente no uniforme. Escogiendo alguna fución de hashing adecuada (Carter y Wegman, 1977), cualquier distribución no uniforme de claves, puede ser convertida en otra cuya distribución es aproximadamente uniforme. Esto es muy difícil de lograr si se desea que el esquema de almacenamiento de las claves sea tal que acepte el efectuar procesos secuenciales con respecto a su orden lexicográfico. En otras palabras, dada una clave cualquiera K_i, el costo de encontrar la clave que precede o sucede a K_i (K_{i-1}, K_{i+1}) en la secuencia natural, es difícil de soportar.

En este documento presentamos una proposición de un esquema de almacenamiento de las claves en orden lexicográfico, mediante hashing extendible. Para ello, se debe escoger una simple función de hashing h tal que si $h(K_1) < h(K_2)$, entonces $K_1 < K_2$ Una forma trivial de lograr esto es usando simplemente el valor de la clave, directamente como función de hashing. Esto sería equivalente un sistema de acceso directo por la clave a almacenar, lo que es difícil de soportar dado que el

tamaño del directorio tendría que ser igual al tamaño del espacio de claves. Además, la agrupación de la información en páginas carecería de sentido.

Mediante la agrupación de claves en páginas, es posible acotar el tamaño del directorio, usando como función de hashing los primeros p bits de la clave. el valor de p dependerá de factores tales como: el tamaño de página a usar, y la distribución del espacio de claves. Las cotas inferior y superior de este espacio no son, en general, posibles de determinar a priori. Sin embargo, los valores máximo y mínimo de las claves, en cualquier instante son datos que si se conocen. Dado que hashing extendible permite modificar dinámicamente el tamaño de directorio, éste se podría ajustar al espacio de direccionamiento de las claves, en cualquier instante.

Para atacar el problema de la no uniformidad del espacio de claves, habría que particionar el espacio de direccionamiento en el directorio, en bloques de tamaño variable. Por ejemplo, en regiones donde se agrupen muchas claves, habría que escoger tamaños de página mayor que en regiones donde haya escasa agrupación de claves. El factor tamaño de página depende básicamente del hardware del computador a usar: tamaño de la pista, número de sectores por pista, etc.

A continuación se propone un esquema de almacenamiento usando hashing extendible, cuya factibilidad se analiza en función de los factores anteriormente señalados. Básicamente, el directorio es administrado en base a los indicadores: p, p'(i) y pm. Como se describió anteriormente, p es la profundidad del directorio en un instante cualquiera, y p'(i) es la profundidad local de la pagina i. El indicador pm representa el número de bits correspondiente a la mayor clave posible de almacenar en la estructura, en un instante cualquiera. Este valor es función de la mayor clave que ha sido almacenada (Km) hasta ese instante, en la forma:

$$pm = \langle \log_2(km) \rangle + 1 \qquad (1)$$

donde, $\langle x \rangle$ es la parte entera de x.

Luego, la mayor clave posible de almacenar en la estructrura es $Kmax = 2^{pm} - 1$. Entonces, para una clave cualquiera k, se usa la función de hashing

$$h(k) = k >> (pm - p) \qquad (2)$$

donde, x >> y es el valor que resulta despues de ajustar x, en y bits hacia la derecha.

Con esta transformación, se cumple con la condición de secuencialidad en el almacenamiento de las claves. Además, para una página i, el valor $2^{pm-p'(i)}$ corresponde al potencial de claves a almacenar en dicha página. Considerando la función (2), con pm=10 y p = 3, la Fig. 4 muestra la distribución de almacenamiento en las páginas, para el mismo esquema de la Fig. 2.

En este caso la estructura, en ese instante, tiene capacidad para almacenar un máximo de 1024 claves (desde la clave 0 hasta la clave 1023). El mínimo número de claves en una página i, se produce cuando p'(i) = p. Esto ocurre con las páginas apuntadas por los punteros 100 y 101; cada una de ellas puede almacenar $2^{(10-3)} = 128$ claves.

Qué pasa si se desea almacenar una clave k que es mayor que la mayor clave (Kmax) posible de almacenar en la estructura? En este caso, se debe duplicar la profundidad del directorio tantas veces como sea la diferencia ($\langle \log_2 k \rangle + 1$) - pm. Los punteros hasta antes de la duplicación, permanecen inalterados; solo se crean nuevos punteros a partir del último (los que señalan a la nueva página). Además, el valor de pm debe incrementarse en la misma cantidad en que fue incrementado p.

Considerando la Fig. 4, supongamos que se desea insertar la clave 3540. En este caso, 3540 > 1023, y como $\langle \log_2 3540 \rangle + 1 - 10$ es igual a 1, la profundidad del directorio y el número de bits de la clave mayor deben aumentarse en 1. La Fig. 5 muestra este nuevo esquema. Nótese que ahora la profundidad local de cada página, disminuyó en uno con respecto a la profundidad del directorio, ya que sólo se agregaron nuevos punteros a la nueva página (p' para ésta página vale cero). Luego, debe llevarse un contador por cada duplicación del directorio en estas circunstancias, para corregir el valor de p'.

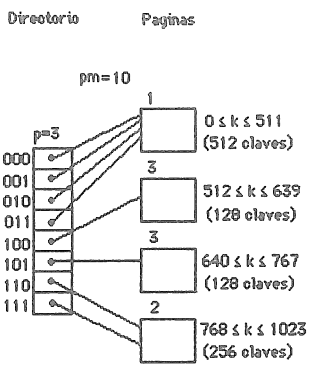


Fig 4 Almacenamiento Secuencial

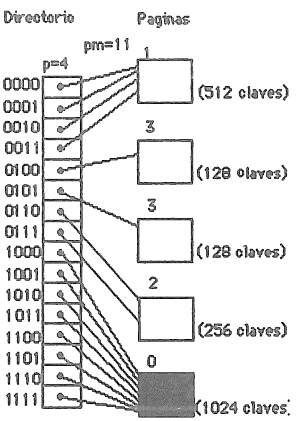


Fig 5 Duplicacion del directorio (K > Kmax)

En la práctica, la duplicación del directorio, del esquema de la Fig 5, podría posponerse, si la página anterior (en el orden lexicográfico de claves) tuviera suficiente capacidad (por ejemplo, su factor de ocupación es menor que 60%). Así, la nueva clave se puede almacenar en dicha página, junto con toda nueva clave a insertar que sea mayor que Kmax (esto debe considerarse en la función de hashing). Si el factor de ocupación de esta página aumenta demasiado, entonces se debe proceder a duplicar el directorio como se indicó anteriormente, distribuyendo las claves mayores que Kmax en una nueva página. De este modo, se evita que valores ocacionales del espacio de claves hagan aumentar el directorio en forma desmedida.

Tal como se mencionó anteriormente, el directorio también se duplica cuando se llena una página cuya profundidad local es igual a p (la profundidad del directorio). En este caso, p se incrementa en 1 pero pm permanece inalterado. Mientras menor sea la diferencia entre pm y p, menor será la cantidad de claves posibles de almacenar en

cada página. En el caso extremo en que pm sea igual a p, el espacio de claves se hace igual al espacio de direccionamiento, y toda página i en que p'(i) = p, limitaría su potencial de claves a 1. Esta situación se produce cuando el espacio de claves tiende a localizarse en torno a un cierto número reducido de páginas. En este caso, además de la no homogeneidad en el factor de ocupación de las páginas, se puede producir un crecimiento desmedido del directorio (por sucesivos rebalses en las páginas). Para evitar este fenómeno, habría que escoger un valor límite (DI) para la diferencia (pm - p). Entonces, al producirse un rebalse en una página i, y ademas se cumple que, p'(i) = p, y (pm-p) = DI, hay que procurar un espacio de almacenamiento para todo nuevo elemento a insertar en dicha página, sin duplicar el directorio.

Una solución podría consistir en ligar otra página a continuación de la página i en cuestión. Esta solución puede conducir a un deterioro importante en el tiempo de acceso promedio a los elementos, si la no uniformidad del espacio de los identificadores es excesiva.

La solución que se propone para evitar el encadenamiento de páginas, consiste en sobredimensionar ligeramente al directorio de modo que éste pueda diferenciar entre páginas de tamaños diferentes. De este modo, al producirse un rebalse en la página i, en las condiciones descritas anteriormente, se traspasa la información de dicha página a una de mayor tamaño, indicando su localización en el directorio (por ejemplo con los primeros d bits de cada puntero, siendo 2^d mayor o igual al número de tamaños distintos de página). Así, se mantiene el tiempo máximo de respuesta al equivalente de dos accesos al disco.

El problema radica en qué y cuántos tamaños de páginas escoger. Con respecto al tamaño de página, hay que considerar que éste es, en general, dependiente del sistema operativo y la configuración de los dispositivos de disco utilizados (tamaño del sector, pista etc.).

Como se mencionó anteriormente, $2^{pm-p'(j)}$ representa el potencial de ítems a almacenar en la página j. Entonces, cuando se cumpla que, DI=(pm-p'(j)) y (p'(j)=p), la información de la página j irá emigrando (por posibles rebalses) a páginas de mayor tamaño hasta un cierto límite superior de tamaño de página (T_{max}). Luego, para permitir la ocupación de esta página y no desperdiciar espacio de almacenamiento, el valor de DI deberá escogerse como el mínimo valor tal que:

$$2^{DI} \geq \langle T_{max} / Ti \rangle \qquad (3)$$

donde Ti es el tamaño de cada ítem a almacenar (asumiendo que es fijo)

Considerando que normalmente el acceso físico al disco es por sector, el tamaño de página inicial (página más pequeña), podría escogerse como el tamaño de un sector (o un submúltiplo de éste). Así, dado que se espera una no uniformidad en el espacio de direccionamiento, se evita un sobredimensionamiento de páginas en regiones con escasa agrupación de ítems. En caso de rebalse en una de estas páginas, su información se puede traspasar a otra página cuyo tamaño podrá ser un múltiplo del tamaño de un sector. Así sucesivamente hasta alcanzar el valor T_{max}.

Para optimizar el tiempo de acceso a memoria secundaria, el almacenamiento de la información en el disco debería ser tal que, cada página, se agrupe en sectores contiguos ("sector interleaving") en una misma pista (Pechura et al,1983). De este modo, mientras se procesa un sector, el siguiente podría ser accesado sin necesidad de esperar una rotación completa del disco. Entonces, podría considerarse como valor aceptable para T_{max}, el valor más cercano al tamaño de la pista (Severance y Duhne,1976).

En cuanto al número de páginas (Np) a escoger, se propone que éste sea igual a

$$Np = \langle \log_2 SPP \rangle \qquad (4)$$

donde SPP es el número de sectores por pista.

Entonces, se escoge como tamaño inicial de página 1 sector del disco, luego 2 sectores, luego 4, etc. Y así sucesivamente hasta Np tamaños. El aumento exponencial del tamaño de página evita la emigración excesiva de información a distintas páginas, en regiones donde se agrupa una gran cantidad de claves. El tamaño inicial de página podría, escogerse algo menor que el tamaño de un sector, asegurando así que aún en zonas de escasa agrupación de claves, se mantenga el factor de ocupación de las páginas por sobre el 50%.

Manteniendo un esquema óptimo de "interleaving" de sectores, se asegura que el tiempo de acceso a una página sea a lo más el equivalente a una rotación del disco. Es razonable esperar que el costo de accesar el directorio sea poco significativo, ya que normalmente estará en memoria principal, aún en aplicaciones que involucren archivos de tamaño relativamente grande (por ejemplo más de 1 millón de registros). Si el directorio no cabe completamente en memoria principal, se puede paginar en bloques de tamaño fijo, manteniendo en memoria principal las páginas de mayor uso.

V Conclusiones.

Se ha propuesto un sistema de almacenamiento y recuperación de información, basado en el esquema de hashing extendible. Nuestro sistema permite almacenar y procesar la información en orden lexicográfico, sin las restricciones de tiempo impuestas por los sistemas de hashing tradicional.

El sistema se basa fundamentalmente en particionar el espacio de almacenamiento en páginas de tamaño variable. La factibilidad del sistema se analiza en base a los requerimientos de almacenamiento en memoria secundaria, y la facilidad para mantener cada página en sectores contiguos en memoria secundaria. Mediante el esquema de particiones se permite, mantener un alto factor de ocupación de las páginas, y acotar el tiempo de acceso a una clave a un máximo de dos accesos al disco.

VI Bibliografía

1. Bayer, R. y McCreigth, E. (1972) Organization and maintenance of large ordered indexes". Acta Informatica Vol 1, No 3, 173-189.

2. Carter, J., y Wegman, M. (1979) Universal Classes of hash functions. Journal of Comput. System Science. Vol 18, No 2, 143-154.

3. Fagin R., Nievergelt, J., Pippenger, N., Strong, H. (1979) Extendible Hashing a fast access method for dynamic files. ACM Transactions on Database Systems, Vol 4, NO 3, 315-344.

4. Knuth, D. E. (1974) The Art of Computer Programming. Vol 3, Addison-Wesley, Reading, Mass.

5. Larson, P. (1978) Dynamic Hashing. BIT 18, 184-201.

6. Litwin, W. (1979) Linear virtual hashing: a new tool for files and tables manipulation. Institut National de Recherche en Informatique et en Automatique, Le Chesnay.

7. Litwin, W. (1980) Trie Hashing. Institut Natinal de Recherche en Informatique et en Automatique, Le Chesnay.

8. Pechura, M. A., Y Schoeffler, J. D. (1983) Estimating file access time of floppy disks. Communication of the ACM, Vol 26 No 10, 754-763.

9. Severance, D., y Duhne, R. (1976) A practitioner's guide to addressing algorithms. Communication of the ACM, Vol 19 No 6, 314-326.

10. Severance, D., y Carlis, J. (1977) A practical approach to selecting record access paths. Computing Surveys, Vol 9, No 4, 259-272.

UN DICCIONARIO DE DATOS PARA UN SISTEMA DE APOYO A LA TOMA DE DECISIONES.

S. ABAD*

RESUMEN

El objetivo de este trabajo es el diseño de un Diccionario de Datos "no estructurado" que funcione como intermediario entre el diseñador de un Sistema de apoyo a la toma de decisiones (DSS: Decision Support System) y su base de datos. El diccionario brinda apoyo en todas las fases del ciclo de vida de la base de datos (análisis, diseño y mantenimiento). La característica de no estructurado lo hace independiente de los modelos conocidos: relacional, jerárquico y de redes.

ABSTRACT

The purpose of this article is to design a "loosely structured" data dictionary to operate as an intermediary between a Decision Support System (DSS) and the DSS data base. This dictionary is a supporting tool throughout all the phases in the database life cycle. The characteristic of being loosely structured makes the dictionary independent from the other known models: relational, hierarchical and network.

* Ingeniero en Computación (USB, 1980), Magister en Ciencias de la Computación (USB, 1983); base de datos, ingeniería de software, sistemas de información, sistemas de operación; Universidad Simón Bolívar - Departamento de Matemáticas y Ciencias de la Computación. Apartado Postal 80659, CARACAS - VENEZUELA. Telfns: 962-1101 al 09 ext. 7191 y 986-4726.

1. INTRODUCCION.

La utilización de los computadores en el desarrollo de aplicaciones ha atravesado varias etapas. En la década de los 50, los computadores se dedicaban al desarrollo de Sistemas EDP (Electronic Data Processing); en los años 60 y 70, el énfasis se hizo en los Sistemas de Información Gerenciales (Management Information Systems). En los años 80 se habla de los Sistemas de Apoyo a la Toma de Decisiones (DSS: Decision Support Systems). Estos últimos surgieron de la combinación de las disciplinas de Investigación de Operaciones y Sistemas de Información, y hoy en día convergen en los DSS, además de las dos areas mencionadas, la de la Inteligencia Artificial, la de Lenguajes y la de Base de Datos. Con los DSS se trata de proveer información a los niveles gerenciales de una empresa, para ayudarlos en su proceso de toma de decisiones, abriéndoles la posibilidad de modelar ellos mismos sus problemas, a través de lenguajes de alto nivel. Una metodología que ha dado buenos resultados en el desarrollo de los DSS, es la del Diseño Incremental y evolutivo. De hecho, hoy en día, tal metodología caracteriza la esencia de los DSS. Se pretende con esta metodología, llevar a cabo todas las fases del desarrollo del sistema (análisis, diseño e implantación) en paralelo. De esta manera, el uso del sistema ayuda a su desarrollo, con lo cual un DSS nunca se culmina, por el contrario, constantemente se le hacen modificaciones para mejorarlo.

Los usuarios finales de los DSS son los gerentes, para que ellos puedan obtener mayor provecho de la información para las decisiones, es vital proveerles una interacción "amigable" y directa con el DSS. Esto se puede lograr diseñando lenguajes de alto nivel, los cuales les permitan expresar clara y precisamente, sus necesidades de información. En este aspecto intervienen las areas de inteligencia artificial y lenguajes de programación.

Por otra parte, los datos que apoyan las decisiones en un DSS pueden ser muy diversos y se requiere de facilidades de recuperación cruzada de información de varias fuentes de datos. En este aspecto son de gran utilidad los Sistemas de Base de Datos. Sin embargo, con el enfoque del diseño evolutivo y el grado de incertidumbre presente en la etapa de análisis, la base de datos de apoyo al DSS, no tiene estructura, es un ente difuso y en crecimiento. En consecuencia, es necesario proveer mecanismos de manejo de esa base de datos "no estructurada". Como complemento de la base de datos, una herramienta muy útil y poderosa para el control y la administración de los datos, es un Diccionario de Datos (DD) (referencias 1, 3 y 8). En el DD de un DSS se debe considerar la no estructuración de los datos que se describen.

La base de datos para un sistema de apoyo a la toma de decisiones, y el diccionario de datos correspondiente, evolucionan a medida que se desarrolla el sistema. En tal evolución se pueden agregar o eliminar elementos del DD y se pueden modificar las interrelaciones entre los mismos. Bajo estas condiciones, el DD ofrece mas facilidades en la medida que dispone de una estructura flexible para adaptarse a los cambios. Tal

flexibilidad requiere de un mecanismo apropiado para recuperar y actualizar información. Motro en [4] propone un método para recuperar información en bases de datos "no estructuradas", llamado "browsing" (hojeada). De acuerdo con este método, se realiza una búsqueda exploratoria sobre la base de datos, sin suponer conocida su organización, mas aún, sin que tenga que existir alguna. Parte de las ideas expresadas por Motro en el artículo mencionado, se utilizan como base para construir el DD propuesto.

En el presente trabajo se describe el diseño de un Diccionario de Datos para un Sistema de apoyo a la toma de decisiones, tomando en cuenta las características de no estructuración de la base de datos de este último. En la sección 2 se introducen los conceptos en los cuales se apoyan el diseño y la construcción del DD; en la sección 3 se define el diccionario propuesto y en la sección 4 se presentan las conclusiones del trabajo y se plantean algunos problemas que requieren investigación.

2. CONCEPTOS BASICOS.

El objetivo de esta sección es plantear los conceptos utilizados en el diseño del diccionario de datos para un DSS.

El punto de partida es que un diccionario de datos es una base de datos con información acerca de los datos de una base de datos. Para diseñar una base de datos se comienza por establecer un modelo conceptual. Existen varias alternativas para modelar conceptualmente los datos ([2], [5] y [6]), entre ellas, una de las mas conocidas y ampliamente utilizada es la del Modelo E-R (Entity-Relationship) ([2], [7]). De acuerdo con este modelo, los datos se representan con entidades (entes distinguibles) y con interrelaciones entre esas entidades. Los conceptos asociados a este modelo son los siguientes:

ENTIDAD — ente distinguible del cual pueden registrarse datos. Puede ser un evento, una persona, un concepto. Contraejemplo: fósforos.

CONJUNTO ENTIDAD — grupo de entidades similares

ATRIBUTO — aquellas propiedades que caracterizan una entidad. También se le llama elemento de dato.

VALOR DE UN ATRIBUTO — dato contenido en un atributo. Puede ser un valor cuantitativo, cualitativo o descriptivo.

DOMINIO — conjunto de valores. Una entidad particular de un conjunto entidad, tiene asociado a cada atributo, un valor tomado de algun dominio.

CLAVE — Atributo o grupo de atributos cuyos valores identifican sin ambiguedad, cada entidad dentro de un conjunto entidad. Como toda entidad es distinguible, cada conjunto entidad tiene por lo menos una clave.

INTERRELACION — Se define una interrelación entre conjuntos entidad como un conjunto

de listas de elementos de los conjuntos entidad, ordenadas. Sean E_1, E_2, ... , E_n los conjuntos entidad bajo la interrelación, los elementos de esta última tendrán la forma general: $(e_1,e_2, \ldots ,e_n)$ donde $e_i \in E_i$ para todo i. Lo más común es que la lista tenga solo dos elementos, pero en general puede tener más de dos.

En el diseño que se propone en este trabajo, se utilizan las entidades para representar las unidades básicas de datos en el diccionario de datos, y se utiliza la noción de hecho ("fact") (definida por Motro en [4]) para representar las interrelaciones entre tales entidades. La definición de hecho es la siguiente:

HECHO es una terna (e_1,r,e_2) donde e_1 y e_2 son entidades y r es el nombre de la interrelación que las asocia.

Si se denomina E al conjunto de todas las entidades e I al conjunto de todos los nombres de interrelación, entonces los hechos son elementos del conjunto (E x I x E). Si en la definición de hecho, e_1 y e_2 son conjuntos entidad, al hecho expresado se le llama CONJUNTO HECHO. Cuando se elige un elemento particular de cada uno de los conjuntos entidad que intervienen en el hecho, se obtiene una instancia particular del conjunto hecho.

El conjunto de hechos que expresan interrelaciones entre entidades (podemos considerar a las interrelaciones como entidades, [2]), se puede particionar en dos conjuntos:

Interrelaciones individuales (H_i): contiene las interrelaciones que caracterizan a un conjunto entidad y que son aplicables a todo elemento de él. Por ejemplo, suponga el hecho: (elem-dato,OCUPA,longitud) donde elem-dato y longitud son conjuntos entidad. Una instancia de este hecho es (cédula,OCUPA,9) donde cédula pertenece al conjunto elem-dato y 9 pertenece al conjunto longitud. Se dice que la interrelación OCUPA es una interrelación individual porque cada elemento de elem-dato, está asociado con un elemento de longitud. En este conjunto de interrelaciones individuales, existen ciertos hechos muy particulares, llamados "hechos totales" los cuales se utilizan para expresar alguna propiedad de los elementos que forman el conjunto entidad. Estos hechos se denotan por (E_i,R,e_m) donde E_i es un conjunto entidad y e_m es una entidad particular. Este hecho se interpreta: todos los elementos de E_i están interrelacionados de acuerdo a R, con el elemento e_m. Por ejemplo, si LONG-9 es un conjunto de todos los elementos de 9 posiciones, el hecho (LONG-9,OCUPA,9) dice que todos los elementos de LONG-9 ocupan 9 posiciones.

Interrelaciones de clase (H_c): contiene las interrelaciones que caracterizan a un conjunto entidad pero que no son aplicables a cada elemento del conjunto sino al todo. Por ejemplo, (elem-dato,N°-TOTAL,500) es un hecho que se interpreta: el número total de elementos de datos incluidos en el DD es 500, entonces se dice que la

interrelación "N°_TOTAL" es una interrelación de clase.

Existen ciertas interrelaciones que expresan condiciones especiales sobre las entidades que asocian. Entre ellas, dos de las más utilizadas son las de pertenencia y las de generalización.

La interrelación de pertenencia expresa que una de las entidades es una instancia de la otra. A los hechos donde ellas participan se les llama "hechos de pertenencia", y se denotan así: $(e_1, \in, e_2)$ donde e_1 es una instancia de e_2. Un ejemplo de este tipo de interrelación se encuentra en el hecho siguiente (M.Contreras, $\in$, estudiante) el cual nos dice que la entidad M.Contreras pertenece al conjunto entidad estudiante.

En el tipo de interrelación de generalización, el sentido expresado por una de las entidades es más general que el expresado por la otra entidad en el hecho. Los hechos donde estas interrelaciones participan se llaman "hechos de generalización" y se denotan así: $(e_1, \subset, e_2)$. Por ejemplo, el hecho (renault, $\subset$, carro) es de generalización y significa que todo elemento del conjunto entidad renault, también es un elemento del conjunto entidad carro, pero lo inverso no es cierto.

A partir de los hechos, cualquiera que sea su tipo, se puede deducir cierta información que no está explícita en los mismos. Por ejemplo, si cédula es un elemento del conjunto LONG-9 y todos los elementos de LONG-9 ocupan 9 posiciones, se puede deducir que cédula ocupa 9 posiciones. Llevando estas afirmaciones a hechos, tendríamos:

(cédula, $\in$, LONG-9) y (LONG-9, OCUPA, 9)

de lo cual deberíamos poder concluir el hecho: (cédula, OCUPA, 9). Este último hecho no tiene por qué existir explícitamente, pues se puede inferir de los dos primeros. Mas aún, la conclusión obtenida se puede generalizar como:

Si (X, $\in$, LONG-9) y (LONG-9, OCUPA, 9) $\Rightarrow$ (X, OCUPA, 9) (1)

donde X es una variable que puede tomar como valor cualquier elemento del conjunto LONG-9. Los hechos donde intervienen variables se les llama "templates" (patrones).

A una expresión del tipo mostrado en (1) se le llama "regla de inferencia". El tipo de deducción que se utilizó en el ejemplo se puede generalizar aún mas con la siguiente regla:

$\forall r \in I$: (s, $\in$, t) y (t, r, u) $\Rightarrow$ (s, r, u)

Existe un conjunto de reglas de inferencia universales ([4]) que se pueden aplicar al ambiente de base de datos. Algunas de estas reglas surgen de hechos de generalización o de pertenencia y son muy utiles, las que se muestran a continuación son algunas de ellas:

$\forall r \in I$: (s,r,t) y (s', $\subset$, s) $\Rightarrow$ (s',r,t)

$\forall r \in I$: (s,r,t) y (r, $\subset$, r') $\Rightarrow$ (s,r',t)

$\forall r \in I$: (s,r,t) y (t, $\subset$, t') $\Rightarrow$ (s,r,t')

$\forall r \in I$: (s,r,t) y (s', $\in$, s) $\Rightarrow$ (s',r,t)

Un aspecto vital en la vida de una base de datos, es la integridad de los datos. A menudo la porción más engorrosa del desarrollo de programas de aplicación y de las operaciones de mantenimiento, es la del control de la integridad de los datos. Es deseable entonces, disponer de un mecanismo automatizado para verificar tal integridad al efectuar operaciones de actualización de la base de datos. Una vía para implantar tal mecanismo es agregar al DD la información concerniente a las restricciones de integridad que es necesario imponer sobre los datos, y que antes de realizar cualquier actualización, se verifique automáticamente con el DD, que tal actualización no viole la integridad de los datos.

Las restricciones de integridad se pueden expresar con reglas como la siguiente: (x, ∈, cedula) ⇒ (x,MAYOR-QUE,0). Esta regla se interpreta como que todo elemento del conjunto de valores válidos para la cédula de identidad, debe ser positivo. Estas restricciones, en lo que respecta a forma, son como las reglas de inferencia.

Es oportuno aclarar que con estas construcciones se pueden expresar hechos contradictorios, por ejemplo, si además de la restricción expresada anteriormente, se incluyera la siguiente regla: (x,∈,cédula) ⇒ (x,MENOR-QUE,0), se crearía una contradicción puesto que todo elemento del conjunto cédula no puede ser mayor y menor que cero simultáneamente. La contradicción provino de que las interrelaciones MAYOR-QUE y MENOR-QUE no pueden establecerse simultáneamente sobre los mismos elementos, porque son contradictorias. En consecuencia, es útil conocer cuales interrelaciones son contradictorias entre sí, para evitar la existencia de hechos contradictorios en la base de datos y en el diccionario de datos.

Durante el diseño de una base de datos, es de suma relevancia conocer la semántica de los datos, para poder construir una base de datos fiel a la realidad. Por ejemplo se tienen los conjuntos entidad tutor y estudiante; se admite que cada tutor pueda tener varios estudiantes, pero cada estudiante tiene a lo sumo un tutor, ello establece una interrelación 1:n de tutor a estudiante, la cual se podría expresar con la siguiente regla:

(x, ∈, tutor) y (e, ∈, estudiante) y
(x,ES-TUTOR,e) y (z, ∈, tutor) ⇒ (x,=,z)
(z,ES-TUTOR,e)

Sin embargo, como las interrelaciones 1:n, junto con las 1:1 y las n:m son muy comunes en el ambiente de base de datos, es muy útil disponer de una expresión abreviada para representarlas. Una manera de hacerlo, es utilizar un operador llamado TIPO, el cual se aplica a las interrelaciones y les asigna su tipo. La forma general de este operador será: TIPO(interrelación) = valor, los valores posibles son: 1:1,1:n, n:1 y n:m. En nuestro ejemplo, se definiría ES-TUTOR como una interrelación 1:n, y se escribiría como: TIPO(ES-TUTOR) = 1:n.

Al utilizar este operador se expresa una restricción de integridad sobre los elementos de los conjuntos: tutor y estudiante. En general, cualquier información a-

cerca de la semántica de los datos, constituye una restricción de integridad, por lo tanto, se puede expresar mediante reglas.

Llegados a este punto, estamos en disposición de definir formalmente las reglas.

Sea V el conjunto de variables entidades y llamemos EV a E ∪ V e IV a I ∪ V, se define REGLA como el par ⟨ L , M⟩, donde L, M ⊂ (EV x IV x EV). Informalmente, se puede utilizar como forma general de una regla, a la siguiente expresión:

expr-1 de hechos ⇒ expr-2 de hechos

(L ⇒ M)

Otro concepto importante, antes de definir lo que se entiende por una base de datos no estructurada, es el concepto de CLAUSURA: Sea H un conjunto de hechos y R un conjunto de reglas, se llama CLAUSURA de H bajo R, al conjunto de hechos que se obtienen de aplicar repetidamente las reglas de R a los hechos en H. Toda clausura de H incluye a H.

Finalmente, podemos definir BASE DE DATOS NO ESTRUCTURADA como una base de datos constituída por un conjunto H de hechos y un conjunto R de reglas, tales que la clausura de H bajo R está libre de contradicciones ([4]).

3. DICCIONARIO DE DATOS PROPUESTO.

Siguiendo los conceptos planteados en la sección 2, el diccionario de datos que se propone para un Sistema de apoyo a la toma de decisiones, es una base de datos no estructurada.

El desarrollo del diccionario comienza con la etapa de análisis del sistema DSS. Durante este primera etapa, al descubrir los datos y su semántica, se determinan las entidades, los atributos y las interrelaciones. Con estos elementos se construyen los hechos del diccionario.

En las interrelaciones, de alguna manera está implícita la semántica de los datos, sin embargo existen otros aspectos que es necesario hacer explícitos, como lo son las dependencias funcionales, las reglas de pertenencia de las entidades a las interrelaciones y las dependencias entre elementos de distintos conjuntos entidad. Esto se puede lograr definiendo reglas, como las expresadas en la sección 2.

Con referencia a las restricciones de integridad ([6]), éstas también deben hacerse explícitas en el diccionario, lo cual se lleva a cabo definiendo otras reglas similares a las semánticas.

El diccionario de datos para un DSS, además de ser no estructurado, tiene ciertas características asociadas a su naturaleza de diccionario ([1,8]). Estas características surgen de que él contiene información descriptiva de los elementos de una base de datos. De acuerdo con ello, se definen algunos hechos particulares, muy útiles en este caso. Estos hechos se aplican a los elementos de datos que se describen en el

diccionario. A continuación se muestran estos hechos:

ORIGEN describe, para un elemento de datos, cual es la fuente que lo provee (por ejemplo: una planilla, una persona, etc.).
Notación: (elem-dato, ORIGEN , fuente)

ES-ATRIBUTO indica que un elemento de dato es atributo de una entidad o interrelación.
Notación: (elem-dato, ES-ATRIBUTO , e)
donde e puede ser el nombre de una entidad o de una interrelación

ES-CLAVE indica que un elemento de datos es clave de acceso de una entidad o una interrelación.
Notación: (elem-dato, ES-CLAVE , e)

PERTENENCIA-I indica como es la pertenencia de las entidades a las interrelaciones ([6] pagns. 70 y 71). Esta pertenencia puede ser de varios tipos:
Opcional: cuando la existencia de las dos entidades participantes no depende de la interrelación.
Contingencia: cuando la existencia de una de las dos entidades es dependiente de la interrelación.
Condicional: cuando la existencia de una de las entidades depende de la interrelación, de acuerdo a un criterio que se puede expresar con una condición booleana. Es un caso particular de la contingencia.
Obligatoria: cuando la existencia de ambas entidades es dependiente de la interrelación.
Notación: (entidad, "tipo-pertenencia" , interrelación)

SINONIMO especifica un sinónimo para un elemento de dato.
Notación: (elem-dato, SINONIMO , elem-dato-1)
donde elem-dato-1 es un sinónimo de elem-dato

Un aspecto muy importante en la descripción de los elementos de datos, que además afecta la integridad, es la definición de los dominios de los atributos. El dominio de un atributo es su conjunto de valores posibles. Los dominios se pueden definir como conjuntos, y luego se especifica cuales atributos pertenecen a cada uno de ellos, a través de un hecho de pertenencia. Por lo tanto, en el diccionario se debe incluir una facilidad para definir conjuntos tanto por extensión como por comprensión.

Consultas.

Para hacer consultas al DD no estructurado, se propone un mecanismo constituído por tres porciones: interrogación, navegación y búsqueda por sinónimos. A continuación se describe cada una de ellas.

Interrogación.

El mecanismo de interrogación a un DD no estructurado, que se propone, utiliza variables y "templates", similares a los construidos para expresar las reglas. Estos dos elementos forman parte de las fórmulas bien formadas que se utilizan para construir las interrogaciones. Una fórmula bien formada está definida por lo siguiente:

1. Los "templates" son los únicos predicados, y cada predicado es una fórmula atómica.
2. Si A y B son fórmulas y X es una variable, entonces $(A \wedge B)$, $(A \vee B)$, $((\forall x)\ A)$ y $((\exists x)\ A)$ son fórmulas.
3. Una expresión es una fórmula si y solo si se puede demostrar que es una fórmula, en base a las cláusulas 1 y 2, anteriores.

En las expresiones $((\forall x)\ A)$ y $((\exists x)\ A)$, se dice que A es el alcance del cuantificador. Se dice que la ocurrencia de una variable X está ligada en una fórmula, si estádentro del alcance de $(\forall x)$ o de $(\exists x)$, o si es la variable de los cuantificadores. En cualquier otro caso, la ocurrencia de la variable es libre en la fórmula. Una tupla $(c_1, c_2, \ldots, c_n)$ satisface una fórmula Q con variables libres $(x_1,x_2,\ldots,x_n)$ si $Q(c_1,c_2,\ldots,c_n)$ se cumple.

Se define un QUERY (interrogación) como una fórmula que se construye con predicados, operaciones de conjunción y disyunción y con los cuantificadores universal y existencial.

Sea $Q(x_1,x_2,\ldots,x_n)$ una fórmula con $x_1, x_2, \ldots, x_n$ como sus variables libres. A Q se le llama QUERY (interrogación) y su valor se denota por $\{Q\}$ y es el conjunto de todas las tuplas $(c_1,c_2,\ldots,c_n)$ tales que $Q(c_1,c_2,\ldots,c_n)$ se cumple.

Una fórmula sin variables libres es una proposición. Este formalismo es la base sobre la cual se apoya el mecanismo de interrogación. Cualquier requerimiento de información se puede expresar a través de él.

Navegación.

Existen situaciones en las cuales el usuario puede no saber qué preguntar o cómo hacerlo. En este caso, se utiliza la técnica de "browsing" ([4]). Como veremos a continuación, esta técnica tiene dos modalidades: navegación propiamente dicha y "probing" (indagación). En la primera modalidad de este tipo de consulta, se supone que el usuario no conoce la estructura de la base de datos. Se examina la "vecindad" de una entidad en particular, se toma otra entidad de allí y se repite el proceso de estudiar su vecindad, hasta que el usuario encuentre lo que busca o decida no buscar mas. Esta modalidad utiliza el tipo de "query" mas sencillo, que es el "template" con una o dos variables libres: (x_1,r,x_2). Además, se utiliza el símbolo "*" para indicar todos los nombres de variables libres.

La modalidad de "probing" de este tipo de consulta, permite que el usuario siga una estrategia de ensayo y error cuando conoce algo sobre la base de datos. El punto

crítico de esta modalidad se encuentra cuando el usuario da un "query", y recibe como respuesta el conjunto vacío, de lo cual, en general, no se obtiene mucha información. Esta modalidad entonces, se basa en las reglas de inferencia por generalización. El usuario inicia su búsqueda con un "query", si éste falla, se intenta con otros "queries" derivados del primero, aplicando las reglas de inferencia por generalización. Al conjunto de "queries" derivados de esta forma, se le llama "conjunto de retracción" de Q, se intenta con cada uno de los elementos de él y se le reportan los éxitos al usuario.

Búsqueda por sinónimos.

Cuando durante el desarrollo de una base de datos intervienen varios usuarios, cada uno de ellos puede tener una visión diferente de los datos y requerir información distinta a partir de ellos. Uno de los objetivos de construir una base de datos, es la integración de las diversas visiones, en una visión consolidada que satisfaga a todos los usuarios. Sin embargo, en este intento, pueden presentarse situaciones conflictivas donde no sea posible poner de acuerdo a los usuarios, en cuanto al nombre de entidades o de elementos de datos. En tales casos, es de gran ayuda, permitir que cada usuario siga llamando a los items por el nombre que considere adecuado, para ello se deben registrar en el DD los sinónimos de cada item, de acuerdo al hecho especial SINONIMO, descrito anteriormente en esta sección. Luego, a la hora de hacer consultas, el diccionario debe proveer una facilidad de búsqueda por todos los sinónimos. En particular, cuando un "query" falle, este método debe analizar todos los "queries" que se puedan obtener a partir del fallido, sustituyendo las entidades, los elementos de datos y las interrelaciones, por los sinónimos que estén definidos para cada uno de ellos, si es que los hay.

Construcción de la base de datos a partir del diccionario.

La utilidad del diccionario de datos en el desarrollo de la base de datos del DSS viene dada, principalmente, por dos aspectos: a) la descripción de los datos y su semántica; b) la construcción de la base de datos. Con lo descrito hasta el momento, se cubre el primer aspecto, ahora conviene definir qué facilidades ofrece el DD con referencia a la construcción de la base de datos.

El uso que se propone para el DD en este trabajo, es que sirva de "filtro" entre quien está diseñando el DSS y la base de datos de éste último. Esto se logra, generando a partir de él, una estructura para la base de datos. La estructura más adecuada para esta situación es la relacional, pues es la más cercana al modelo conceptual que sirvió de base al DD. En otras palabras, es la estructura más flexible de las tres mas conocidas (relacional, jerárquica y de redes) ([7]). De esta forma, la estructura se genera cuando el diseñador conozca suficiente sobre los datos como para ensayar una estructura con los datos reales. Luego de creada la base de datos, se realiza una carga real. Las consultas y actualizaciones posteriores se canalizan a través del DD no estructurado, el cual debe traducir dichos requerimientos, en operaciones sobre la base de datos. La

información referente a los metadatos, se consulta directamente en el diccionario.

Las labores de mantenimiento del DD, se llevan a cabo a través de una colección de operadores. Algunos de los operadores más importantes son: ESCRIBA, para agregar hechos al diccionario, a cada hecho se le asocia una clave de identificación; ELIMINE, para eliminar un hecho del diccionario, para especificarlo se puede escribir el hecho, o dar su clave de identificación; INCLUYA, para incluir reglas al diccionario, se le asocia una clave de identificación a la regla, y EXCLUYA, para eliminar una regla del diccionario, para especificarla se escribe o se da su clave de identificación.

Como complemento a los operadores, se incluye una facilidad de consulta al diccionario. Esta facilidad provee toda la meta información de la cual se disponga, para cada entidad y cada interrelación. Por ejemplo, para las entidades se puede consultar en cuáles hechos participa y cuáles son sus atributos; para cada interrelación se puede consultar cuál es su tipo, y como es la pertenencia de las entidades que asocia, a la misma.

A continuación se muestra un ejemplo muy simple, donde se ilustran algunas de las características descritas del diccionario propuesto. Suponga un ambiente académico, para el cual se desea desarrollar una base de datos, para tal fin se definen como entidades: profesor, con atributos nombre, cédula, categoría, teléfono y oficina; estudiante, con atributos nombre-e, cedula-e, n°carnet, carrera; y tesis con los atributos código y título. Entre estas entidades se establecen tres interrelaciones: TUTOR (profesor y estudiante), DIRIGE (profesor y tesis) y DESARROLLA (estudiante y tesis). Para definir esta base de datos en el diccionario propuesto se definirían conjuntos, hechos y reglas. En primer lugar se definirían los conjuntos ENTIDADES e INTERRELACIONES, con los elementos (profesor, estudiante, tesis) y (TUTOR,DIRIGE,DESARROLLA), respectivamente. Entre los hechos se definirían: (nombre,ES-ATRIBUTO,profesor), (cédula,ES-atributo,profesor), (categoría,ES-ATRIBUTO,profesor), (teléfono,ES-ATRIBUTO,profesor), (nombre-e,ES-ATRIBUTO estudiante), (cedula-e,ES-ATRIBUTO,estudiante), (n°carnet,ES-ATRIBUTO,estudiante), (carrera,ES-ATRIBUTO,estudiante), (código,ES-ATRIBUTO,tesis) y (título,ES-ATRIBUTO,tesis) para definir los atributos de cada entidad; (profesor,DIRIGE,tesis), (estudiante, DESARROLLA,tesis) y (profesor,TUTOR,estudiante) para definir las entidades que asocia cada interrelación; tipo(DIRIGE)=1:n, tipo(DESARROLLA)=n:1 y tipo(TUTOR)=1:n para definir los tipos de las interrelaciones; y (tesis,contingencia,DIRIGE), (profesor,opcional, DIRIGE), (estudiante,opcional,DESARROLLA), (tesis,contingencia,DESARROLLA), (profesor, opcional,TUTOR) y (estudiante,opcional,TUTOR) para definir los tipos de pertenencia a las interrelaciones. Seguidamente se muestran dos de las reglas que cabe definir para este caso:

1. (X, $\in$,estudiante) y (T, $\in$,tesis) y (X,DESARROLLA,T) y ($\exists$ w) (w,$\in$,estudiante) y (w,DESARROLLA,T) $\Rightarrow$ (w,=,X)

2. (T,$\in$,tesis) $\Rightarrow$ ($\exists$ w) (w,$\in$,estudiante) y (w,DESARROLLA,T) y

$(\exists x)$ (x,∈,profesor) y (x,DIRIGE,T)

Nota: esta regla expresa la situación de que la pertenencia de tesis a las interrelaciones DIRIGE y DESARROLLA, es de contingencia.

4. CONCLUSIONES.

Actualmente se está trabajando en la implantación del diccionario de datos propuesto. En esta labor se está poniendo especial énfasis en los aspectos:

1. Búsqueda de una estructura de datos adecuada para almacenar los hechos y las reglas del diccionario. Tal estructura debe proveer accesos rápidos.
2. Lenguaje del diccionario. Este lenguaje debe ser amistoso ("user friendly") y cómodo, tanto en la definición como en la consulta y el mantenimiento del diccionario.
3. Interfase con la base de datos del DSS. En este aspecto deben considerarse dos puntos: a) cómo generar una estructura relacional para la base de datos del DSS, a partir de la meta información contenida en el diccionario de datos del mismo, y b) cómo traducir requerimientos expresados de acuerdo a los mecanismos de consulta del diccionario, en operaciones sobre la base de datos que respondan efectivamente a las preguntas y como efectuar esas operaciones.
4. Interfase con un manejador de base de datos (DBMS, data base management system). Este aspecto involucra elegir uno (o más) DBMS desde el cual poder utilizar las facilidades del diccionario de datos propuesto. Esto, por supuesto, tendrá un gran impacto sobre el aspecto 3, mencionado anteriormente.

El servicio que preste el diccionario de datos al sistema de apoyo a la toma de decisiones, se podría ampliar incluyendo facilidades de análisis y diseño de sistemas de información, con herramientas gráficas.

BIBLIOGRAFIA.

[1] ALLEN, F.W.; LOOMIS, S. y MANNINO, M. "The integrated Dictionary/Directory System". Computing Surveys ACM, Vol. 14, N° 2, Junio 1982.

[2] CHEN, P. "The Entity-Relationship Approach to Logical Data Base Design" Q.E.D. Monograph Series on Data Base Management, N° 6, Mass. 1977.

[3] CURTICE, R.M. "DATA DICTIONARIES: An assessment of current practice and problems" IEEE 1981.

[4] MOTRO, A. "Browsing in a loosely structured Database" Communications ACM, January 1984.

[5] PETERS, L.J. "Software Design: Methods and Techniques" Yourdon Press N.Y. 1981.

[6] TEOREY, T.J. y FRY, J.P. "Design of Database structures" Prentice-Hall N.J. 1982.

[7] ULLMAN, J.D. "Principles of Database Systems" Computer Science Press - 1982.

[8] VAN DUYN, J. "Developing a Data Dictionary System" Prentice-Hall - N.J. 1982.

PROJETO DO ESQUEMA CONCEITUAL PARA SISTEMAS DE INFORMAÇÃO: UMA ABORDAGEM USANDO A METODOLOGIA THM.

C.L.G. MEDEIROS e U. SCHIEL

SUMÁRIO

É apresentada a primeira das duas fases de uma metodologia para o projeto de esquemas conceituais para sistemas de informação(SI). Esta metodologia modela tanto os aspectos estáticos como os dinâmicos do SI, com ênfase especial nos últimos. Na primeira fase é desenvolvido um Modelo de Requisitos para descrever o SI através de Grafos Estruturais e de Comportamento baseados em modelo (restrito) semântico de dados.
A metodologia está sendo aplicada com sucesso nodesenvolvimento de um SI para uma pequena biblioteca. Como ilustração, é apresentado o Modelo de Requisitos desse sistema.

ABSTRACT

The first of two plases of a methodology for designing conceptual Schemes for Information Systems(IS) is presented. This methodology models both the static and the dynamic aspects os IS, with special emphasis on the latter. The first plase develops a Requeriments Model which describes the IS by using Behaviour and Structural Graphs, both based on a restricted semantic data model. The methodology is sucessfully being applied in the development of an IS for a small library. As an illustration, the Requirements Model for such system is developed and presented.

Camilo de Lelis Gondim Medeiros,
Tecnólogo em Processamento de Dados-UFPb/Fev/77, atualmente elaborando Dissertação de Mestrado em Sistemas e Computação e Professor Assistente do DSC/UFPb. Áreas de Interesse: Banco de Dados, Sistemas de Informação, Linguagens de Programação, Compiladores. UFPb/DSC/Av. Aprígio Veloso, 882/Bodocongó, 58.100/Campina Grande/Pb. Tel. (083)7222 - Ramal: 510.

Ulrich Schiel,
Bacharel em Matemática-Mackenzie/SP.(1971), Aperfeiçoamento em Matemática/Impa/RJ.(1974). Mestrado em Informática/PUC/RJ.(1977), Doutorado em Informática/Universidade de Stuttgart Alemanha(1984), Professor do DSC/UFPb. Áreas de interesse: Banco de Dados, Sistemas de Informação. UFPb/DSC/Av. APrígio Veloso, 882/Bodocongó, 58.100/Campina Grande/Pb.
Tel. (083)7222 - Ramal: 510.

1. INTRODUÇÃO

Durante os últimos anos foram desenvolvidas várias abordagens para modelagem de sistemas de Informação (SI) (compare com |9|). Todas elas focalizam o problema de derivar um esquema conceitual representando o universo de discurso que pode ser manuseado pelo SI. Para derivar tal esquema conceitual é requerida uma metodologia de projeto para dar suporte tanto ao especialista de aplicação quanto ao Administrador de Banco de Dados (ABD) durante o processo de mapeamento. Ao desenvolver uma metodologia de projeto, deve ser encontrado um equilíbrio entre as necessidades do especialista de aplicação e o ABD. Para dar suporte ao especialista de aplicação durante o processo de estabelecer os requisitos para o SI a metodologia tem de oferecer conceitos orientados mais para a aplicação do que para o banco de dados. Uma vez que o especialista de aplicação é um expert na área de aplicação que deve ser coberta pelo SI, e não um expert em bancos de dados. Por outro lado, o ABD está interessado primariamente nos conceitos de modelagem orientados para banco de dados para descrever os requisitos de SI a ser desenvolvido, para ter uma base bem estabelecida para derivar o esquema conceitual. Para evitar misturas entre o especialista de aplicação e o ABD, é muito importante prover conceitos de modelagem na metodologia de projeto que possam ser usados como uma base de comunicação entre o especialista de aplicação e o ABD.

Para solucionar este conflito Horndasch et.al.(|7|) propuseram uma metodologia de projeto que será chamada de metodologia THM. Esta metodologia será apresentada no presente trabalho com algumas pequenas melhorias e ilustramos sua utilização para a modelagem de um sistema de controle do acervo bibliográfico de uma biblioteca setorial. Por limitações de espaço o sistema não será descrito até os últimos detalhes, mas sim, com detalhes suficientes para ilustrar a metodologia.

A metodologia THM é dividida em duas fases (veja a figura 1):

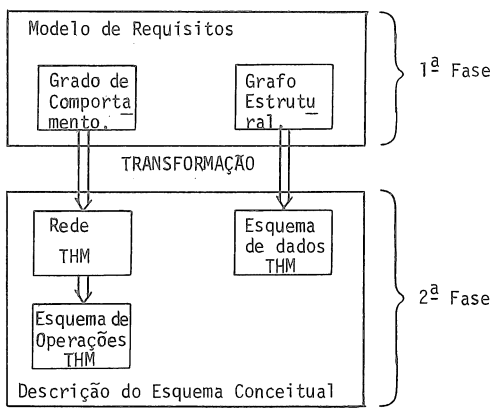


Figura 1: Visão da abordagem em duas fases.

(1) A primeira fase é orientada para a análise dos requisitos para o SI e estabelecimento de um modelo inicial para o SI. Uma vez que os aspectos dinâmicos de concorrência são muito decisivos em um SI, por exemplo quando ele é usado em um ambiente de escritório (veja |2|), é dada ênfase a estes aspectos dinâmicos durante a primeira fase do projeto. Entretanto, desde que o objetivo máximo da metodologia é chegar a uma descrição do esquema conceitual baseada em conceitos do modelo semântico de dados, o processo de análise efetuado pelo especialista de aplicações deve ser guiado também por estes conceitos dos modelos semânticos de dados. Usando diversos passos de refinamento a primeira fase resulta no que é chamado de Modelo de Requisitos (MR) que é composto de duas partes: um Grafo de Comportamento(GC), especificando os eventos induzidos pelo comportamento funcional do SI e o fluxo de informações entre funções ao usar uma rede canal/agência generalizada (compare com |12|) e um Grafo Estrutural(GE) representando a estrutura das informações manipuladas pelo SI. O GE é construído usando um modelo semântico de dados restrito a relações binárias.

(2) Durante a segunda fase o MR é transformado em uma rede de alto nível (chamada Rede THM) provendo um modelo conceitual totalmente formal do SI. Na rede THM o modelo semântico de dados THM (Modelo Temporal - Hierárquico) (|14|, |15|) é usado para descrever os dados que são consumidos/produzidos pelas transições na rede THM. Entre os vários conceitos de Petri-nets foram selecionados conceitos de redes de predicado/transição (PrT-net)(|13|) e modificados apropriadamente para integrar os conceitos de Petri-nets com os conceitos de THM. Em adição à rede THM, é provida uma descrição estática do esquema conceitual descrevendo a estrutura dos dados manipulados pela rede THM.

Têm sido desenvolvidas várias metodologias para a modelagem e projeto dos sistemas de informação. Em |8| é proposta a metodologia ISAC. Entretanto, os grafos de estrutura usados na ISAC não fornecem meios para prover uma especificação detalhada das condições de ativação nem cobrem aspectos de concorrência.

A abordagem descrita em |5| é totalmente descritiva e não inclui quaisquer aspectos operacionais. A metodologia THM tem algumas similaridades com a abordagem introduzida em |10| e |12| que usa as chamadas IML - PrT-nets inscritas. Entretanto, ela difere desta abordagem em diversos aspectos importantes:

(i) Em oposição ao MR de THM a rede canal/agência, usada como um primeiro modelo na abordagem IML, não inclui quaisquer conceitos de modelagem de dados e simplesmente oferece um canal como uma unidade funcional que armazena informação.

(ii) A inscrição IML de PrT-nets é baseada nos Conceitos de Gerenciamento de Informação (CGI) |11| que não provêem a poderosa modelagem (p. ex. os princípios de abstração) dos modernos modelos semânticos de dados como p. ex. no modelo de dados THM, que é usado na metodologia.

(iii) Desde que os aspectos de dados da metodologia THM são modelados em ambas as fases do projeto, é obtido umd esenvolvimento mais paralelo de dados e atividades.

(iv) Em |12| faltam orientações precisas como transformar o primeiro modelo, o modelo canal/agência, no segundo modelo, o IML - PrT-net inscrito. Na metodologia THM são dadas regras para uma transformação sistemática do MR na rede THM.

As partes subsequentes do presente trabalho são organizadas como segue. Na seção 2 é apresentada uma breve introdução ao Modelo de Dados Temporal-Hierárquico (THM). Na seção 3 é introduzido o Modelo de Requisitos consistindo do Grafo de Comportamento e do Grafo Estrutural. Adicionalmente, descreve-se os diferentes passos de desenvolvimento que são usados para gerar o MR. Para propósitos de ilustração, é usado o problema de automação das atividades de uma pequena biblioteca. Em um artigo futuro, proveremos a definição da nossa rede de alto nível, i.e., a rede THM para o sistema citado e discutiremos as regras para transformação do MR em rede THM e a descrição da estrutura de dados THM.

2. O MODELO DE DADOS TEMPORAL-HIERÁRQUICO (THM)

Desde que a metodologia de projeto apresentada resulta em esquemas baseados no modelo semântico de dados THM, é dada uma breve descrição dos conceitos mais importantes de THM. Uma descrição detalhada e também uma formalização podem ser encontradas em |14| e |15|. Em THM tanto os conceitos de modelagem bem-conhecidos de diversos modelos semânticos de dados, p. ex. SDM (|6|) e SHM + (|1|), quanto novos conceitos como p. ex. conceitos de modelagem de tempo, são integrados em uma abordagem consistente.

A parte estática de THM, o esquema de dados, é baseada nos conceitos "entidade", "classe (de entidade)", "relação (binária)". São distinguidos três tipos de relação: (a) relações membro-a-membro descrevendo relações entre membros de duas classes, (b) relações classe-membro especificando relações entre uma classe como um todo e membros de outra classe, e (c) relações de classes descrevendo relações entre duas classes. Para ser capaz de estabelecer uma hierarquia de classes de entidades são incluídos os três princípios de abstração: generalização, agregação e agrupamento em THM. Uma vez que a ênfase é dada aos aspectos de modelagem de tempo, THM não só provê conceitos para descrever o estado atual de entidades e relações, mas também, oferece meios de especificar o tempo de vida de entidades e de relacionamentos, armazenando seu passado e indicando o seu futuro.

A parte dinâmica de THM, o esquema de operações, oferece conceitos para descrever operações que podem ser usadas para transformar um estado do banco de dados em um estado sucessor. Se o sucesso da execução de uma operação depende de requisitos que devem ser atendidos pelo estado atual da base de dados e/ou seu estado sucessor, são incluídas pré-condições e pós-condições respectivamente, na definição de uma operação. Uma operação é construída de operações primitivas (inserir, deletar, para entidades; estabelecer, remover, para relações) e operações complexas já definidas. Para descrever modificações no estado do banco de dados que requerem a execução de operações posteriores, o conceito de evento/partida de THM pode ser usado. Um evento ou depende de uma operação ou de uma condição de tempo. Uma partida (trigger) descreve as ações que devem ser levadas a cabo se ocorrer o ou/os evento(s) associado(s). Finalmente para garantir a integridade semântica do banco de da-

dos relativamente à hierarquia das classes e também relativo às relações e suas inversas, são introduzidos em THM os chamados efeitos colaterais. Eles descrevem as operações que devem ser executadas como uma consequência de se ter executado outras operações.

3. O MODELO DE REQUISITOS

A primeira fase da metodologia de projeto THM pretende derivar o Modelo de Requisitos(MR), que pode ser considerado como uma primeira formalização do SI a ser desenvolvido. O MR será derivado de uma descrição informal dos requisitos do SI usando diversos ciclos de iteração como mostrado na seção 3.3. De nosso ponto de vista é importante considerar tanto os aspectos dinâmicos quanto os estáticos de um SI desde o início uma vez que o projeto do esquema estrutural será imediatamente influenciado pelos aspectos dinâmicos do SI e vice-versa. Portanto, o MR compreende dois tipos diferentes de grafos para cobrir os aspectos estáticos e dinâmicos. Uma vez que o MR será usado na segunda fase do projeto como a base para derivar a rede THM, os conceitos de modelagem oferecidos no MR são orientados para os conceitos de modelagem estáticos e dinâmicos definidos em THM, o modelo semântico de dados que será usado finalmente para representar o esquema conceitual do SI.

O MR é composto de dois grafos o Grafo de Comportamento(GC) e o Grafo Estrutural(GE). Posteriormente, discute-se em detalhes os conceitos de modelagem providos por estas estruturas de grafos.

3.1. O Grafo de Compostamento(GC)

O GC provê uma descrição funcional do SI, especificando o fluxo de informações entre funções e as chamadas unidades de informação (campare com SADT (|13|)). Adicionalmente, o GC descreve o compotamento de cada função por meio das pré e pós-condições, introduz condições de ocorrência para eventos e especifica restrições de integridade para as unidades de informação.

Seu propósito é prover uma visão das funções necessárias no SI e suas semânticas e das informações manipuladas pelas funções.

Consideremos a definição do GC:

Definição 3.1:

Um Grafo de Comportamento(GC) é definido como segue:

Um GC é uma rede dirigida (F,I,E,IF) onde: (1) F é o conjunto das funções,
(2) I é o conjunto de unidades de informação,
(3) E é o conjunto de eventos,
(4) $IF \subseteq ((I \times F) \cup (F \times I) \cup ((E \times F) \cup (F \times E))$ é uma relação representando os arcos do GC.

(1) Funções:

O conjunto de funções F é composto de dois subconjuntos disjuntos: $F = EF \cup CF$, onde

(i) EF é o conjunto das funções elementares,

(ii) CF é o conjunto das funções complexas. Uma função complexa é composta de funções elementares e/ou outras funções complexas. Ao introduzir o conceito de funções elementares e complexas, um GC pode ser decomposto em uma hierarquia de diagramas, i. e, uma função complexa de nível i pode ser representada por um GC separado de nível $i+1$.

(2) Unidades de Informação:

O conjunto de Unidades de informação I é composto de dois subconjuntos disjuntos: I = EXI ∪ INI, onde

(i) EXI é o conjunto de unidades externas de informação descrevendo informações que não serão mantidas no SI, i. e., entrada para o SI e saída do SI.

(ii) INI é o conjunto de unidades internas de informação que serão parte do SI e posteriormente transformadas em classe de entidades na descrição do esquena conceitual. Para uma dada função $f \in F$ o conjunto de unidades de informação de entrada é definido por $pre(f) = \{i \mid (i,f) \in ((IXF) \cap IF)\}$, o conjunto das unidades de informação de saída por: $pos(f) = \{i \mid (f,i) \in ((FXI) \cap IF)$.

(3) Eventos:

O conjunto de eventos é composto de dois subconjuntos disjuntos: E = EE ∪ IE, onde

(i) EE é o conjunto de eventos externos que representam as trocas de estado do mundo real que resultam em uma troca de estado do SI. A ocorrência dos eventos externos é dependente das condições fora do escopo formal do SI.

(ii) IE é o conjunto dos eventos internos. Um evento interno causa uma troca de estado do SI e sua ocorrência depende ou de uma outra troca de estado (gerada por uma função) ou de um sistema manuseado por condições de tempo.

(4) Arcos:

Arcos conectando funções e unidades de informação especificam o fluxo de informações no GC. Um arco, que é um elemento de (ExF), conecta uma função com o evento que especifica sua condição de ativação. Um arco que é um elemento de (FxE), representa o fato de que o evento associado com a função será ativado ao executar a função.

(5) Inscrições:

Unidades de informação, eventos e atividades podem ser inscritos ou por nomes, descrevendo informalmente o significado dos diferentes elementos do GC, ou por um texto descrevendo as condições que devem ser atingidas por eles. Condições para unidades de informação e eventos são chamadas de restrições e ou especificam restrições para as entidades das unidades de informação ou descrevem condições de ocorrência para eventos. Condições para atividades são descritas por pré e pós-condições, respectivamente, especificando condições para as entidades das unidades de informação de entrada e das

unidades de informação de saída, respectivamente. As condições são coletadas em uma lista de condições que é referenciada a partir das funções correspondentes, unidades de informação, e eventos do GC.

(6) Comportamento:

Uma função $f \in F$ consome/ler elementos (entidades) de (alguma de) suas unidades de informação de entrada e entrega elementos (entidades) para (alguma de) suas unidades de informação de saída. As restrições que devem ser mantidas para as entidades das unidades de informação de entrada (saída) são especificadas pelas pre-/pós-condições da função.

Se uma função $f \in F$ está associada com um evento por um arco que é um elemento de (ExF), a ocorrência do evento é um pre-requisito para a execução da função. Se um evento está conectado a uma função $f \in F$ por um arco que é um elemento de (FxE) o evento é gerado tão logo a função tenha sido executada.

Do mesmo modo que com uma rede canal/agência, o GC não provê uma especificação única com relação ao consumo e geração de entidades durante a execução de uma função:

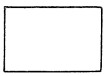
(i) nem todas as unidades de informação de entrada devem fornecer entidades;

(ii) nem todas as unidades de informação de saída recebem unidades;

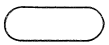
(iii) os arcos entre unidades de informação e funções não especificam se as entidades são consumidas, modificadas ou simplesmente lidas;

(iv) os arcos não especificam o número de entidades que são consumidas ou produzidas.

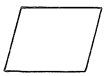
Na construção do GC será usada a seguinte notação:



Para funções.



Para unidades internas de informação.

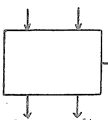


Para unidades externas de informação.

Para eventos.

Para arcos (fluxo de informações).

Para entrada ou saída de informações mutuamente exclusivas.



Um retângulo envolvendo funções, eventos..., para representar o detalhamento de uma função "A".

3.2. O Grafo Estrutural(GE)

Para modelar os aspectos estáticos de um SI o GE é usado no MR. O GE baseia-se em um modelo semântico de dados oferecendo um subconjunto dos conceitos de modelagem definidos em THM. Uma vez que THM será usado para a descrição do esquema conceitual (na fase 2), o processo de análise efetuado na primeira fase é dirigido para os conceitos finais de modelagem desde o início. A principal restrição é que os princípios de abstração de THM (generalização, agregação, agrupamento) não foram incluídos no GE. A razão para tal restrição do modelo semântico de dados na primeira fase do projeto é que o especialista de aplicação não está familiarizado com todos os diferentes conceitos de modelagem usualmente oferecidos por tal modelo. Além disso não é prático desenvolver uma descrição de um esquema conceitual completo em uma fase de projeto.

O GE é desenvolvido através da análise das unidades de informação do GC como também das restrições e pré-/pós-condições definidas no GC. Durante este processo de derivação são definidas as classes de entidades e são estabelecidas as relações entre as classes de entidades. As restrições requeridas para as classes de entidades do GE são adicionadas à lista de condições já introduzidas no GC.

A seguir é apresentada uma definição do GE, i. e., o modelo semântico de dados restrito. Entretanto, não será apresentada uma DDL uma vez que pode ser usado diretamente o subconjunto de THM/DDL (|14| e |15|) requerido para nosso modelo semântico de dados restrito.

Definição 3.2.

Um Grafo Estrutural(GE) é definido como segue:

Um GE é um grafo dirigido (CE;R)

(i) CE é o conjunto de classes de entidades,

(ii) R é o conjunto de relações binárias entre classes de entidades representando os arcos do GE.

(1) Classes de Entidades:

Uma entidade representa um objeto do mundo real. Um conjunto de entidades que têm as mesmas propriedades é uma classe de entidades.

(2) Relações:

O conjunto das relações é composto de três subconjuntos disjuntos:
$R = CCR \cup CMR \cup MMR$, onde:

(i) CCR é o conjunto de relações classe-a-classe,
(ii) CMR é o conjunto de relações classe-membro,
(iii) MMR é o conjunto de relações membro-a-membro:

Para representar uma relação $\mathcal{R}$ entre duas classes C1 e C2 é usada a notação 'C1 $\mathcal{R}$ C2'.

(3) Restrições:

(i) Todas as classes de entidades devem ter nomes únicos;

(ii) Todas as relações devem ter nomes diferentes;

(iii) A especificação de relações classe-a-membro e membro-a-membro deve incluir uma especificação de cardinalidade c = (mim,max). Para uma dada relação entre duas classes A e B (mim,max) especifica o número mínimo e o número máximo de entidades da classe B relacionadas a uma entidade de A. ('*' representa um número indeterminado, 'L' representa um limite previamente estabelecido.

É oferecido um conjunto de relações predefinidas que pode ser usado posteriormente para descrever as hierarquias de abstração requeridas para a descrição do esquema conceitual.

A seguir, são apresentadas as relações "default" para relações membro-a-membro.

Relações membro-a-membro:

Relação	Cardinalidade	Descrição
1) C1 é um C2	(1,1)	.Todo elemento de C1 tem que ser elemento de C2.
C1 é um C2	(0,1)	.Um elemento de C1 pode ser um elemento de C2.
2) C1 é parte de C2	(1,1)	.Um elemento *X* de C1 é sempre parte de exatamente um elmento *Y* de C2.
C1 é parte de C2	(0,1)	.Um elemento *X* de C1 pode ser parte de um elemento *Y* de C2.
(Para max > 1 tem-se casos análogos).		
3) C2 tem-comp. C1	(1,1)	.Um elemento *Y* de C2 sempre inclui exatamente um componente *X* de C1.
(Para max > 1 tem-se situação análoga 'max = *' e também 'min = 0' não devem ocorrer).		
4) C1 é elemento de C2	(1,1)	.Uma entidade *X* de C1 é sempre um elemento de exatamente uma entidade de C2.
(Todos os outros tipos de cardinalidade são também permitidos).		
5) C2 tem-elemento C1	(1,*n*)	.Uma entidade *Y* de C2 tem *n* entidades X1, X2...,X*n* de C1 como elementos.
('min = 0' é permitido. '*n*>1' tem que ser mantido)		

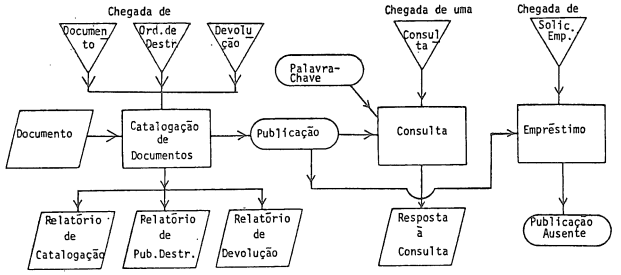


Figura 2a: GC - Fluxo de Informações do Sistema de Controle de Acervo Bibliográfico

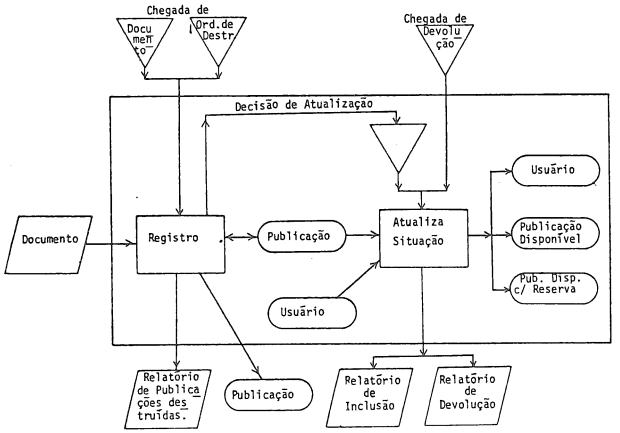


Figura 2b: GC - Fluxo de Informações de Catalogação de Documentos

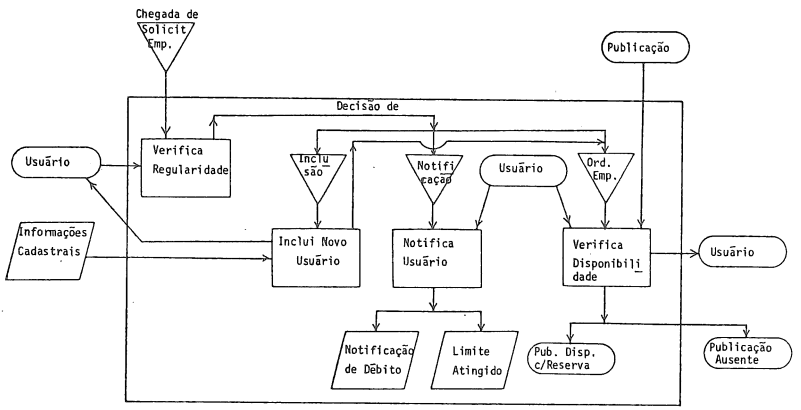


Figura 2c: GC - Fluxo de Informações de Empréstimo.

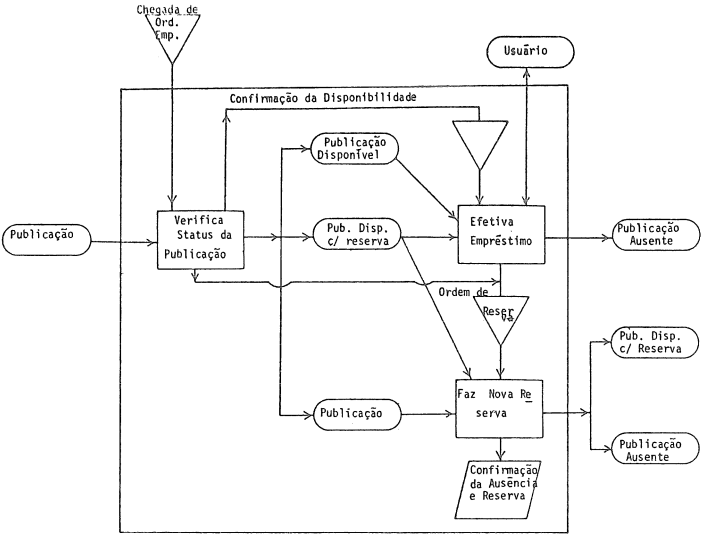


Figura 2d: GC - Fluxo de Informação de Verifica_ Disponibilidade

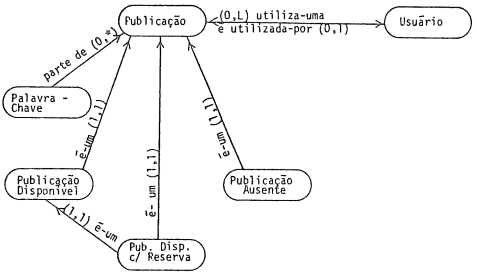


Figura 3: Grafo Estrutural

CLASSE/RELAÇÃO \ FUNÇÃO	CONSULTA	ATUALIZA SITUAÇÃO	EFETIVA EMPRÉSTIMO
Publicação	L		
Usuário		M	*
Publicação Ausente			G
Publicação Disponível		G	C
Publicação Disp. c/Reserva		G	C
Palavra Chave	L		

Figura 4: Parte da Tabela de Acessos para o GC da Figura 2.

3.3. Os passos para desenvolver MR

Como foi mostrado na introdução a metodologia de projeto é dividida em duas fases. Nesta seção serão discutidos os vários passos que devem ser efetuados durante a primeira fase para o desenvolvimento do MR. No final desta seção serão ilustrados os diferentes passos com a discussão do problema de controle do acervo bibliográfico de uma pequena biblioteca. A suposição básica é que não será assumida uma estratégia estritamente Top-down. Ao contrário, assume-se que as estruturas dos grafos do MR são desenvolvidas de maneira iterativa, i.e., modificar uma das estruturas de grafo pode resultar em modificações de outras estruturas de grafo.

Passo 1: O Desenvolvimento do GC

Dependendo dos passos anteriores, os seguintes casos podem ser distinguidos durante o desenvolvimento do GC:

a) É desenvolvida uma primeira visão do universo de discurso, analizando o fluxo de informações entre funções e determinando os eventos que ativam as funções.

b) Um GC existente é extendido com a inclusão de novas funções, unidades de informação, e eventos.

c) Unidades de informação de entrada/saída que ainda não foram consideradas são conectadas com as funções correspondentes.

d) Uma função complexa é decomposta em diversas subfunções conectados por arcos de fluxo de informação. Este processo de decomposição é continuado enquanto a análise do fluxo de informações resulte na identificação de novas subfunções (veja figura 2).

e) As atividades do GC são analisadas relativamente às pré/pós-condições que têm que existir para uma execução satisfatória das atividades. Ou as atividades são inscritas pelas pré/pós-condições ou elas são inseridas na lista de condições (figura 5) que é então referenciada no GC.

f) As unidades de informação são especificadas posteriormente com a inserção de restrições na lista de condições para garantir a integridade semântica das informações mantidas nestas unidades (veja figura 5).

g) A descrição dos eventos é desenvolvida posteriormente com a especificação das condições de ocorrência para os eventos. Estas condições de ocorrências são também registradas na lista de condições.

Passo 2: A Derivação do Grafo Estrutural(GE)

Ao derivar o GE as unidades de informação e também a lista de condições do GC devem ser analisadas. Deve ser verificado como as informações descritas pelas unidades de informação e as restrições do GC podem ser representadas por classes de entidades e relações. Adicionalmente, deve ser verificado que restrições da lista de condições são cobertas pelas relações definidas entre as classes de entidades. Se é introduzida uma nova relação, tem

FUNÇÃO	PRÉ/PÓS-CONDIÇÕES	
CATALOGAÇÃO DE DOCUMENTOS.	Pré-Cond. Pós-Cond.	Existência ou destruição de um documento.
CONSULTA	Pré-Cond. Pós-Cond.	
EMPRÉSTIMO	Pré-Cond. Pós-Cond.	Existência da publicação.
REGISTRO	Pré-Cond. Pós-Cond.	Registro de documento não existente ou destruição de publicação que já existe. Ter sido efetuado o registro da publicação (cadastramento ou cancelamento).
ATUALIZAÇÃO SITUAÇÃO	Pré-Cond. Pós-Cond.	A publicação fica disponível ou reservada e o total de empréstimos do usuário é atualizado.
VERIFICA REGULARIDADE	Pré-Cond. Pós-Cond.	Existência do usuário.
INCLUI NOVO USUÁRIO	Pré-Cond. Pós Cond.	Não existência do usuário. Ter sido efetuada a inclusão do novo usuário.
NOTIFICA USUÁRIO	Pré Cond. Pós-Cond.	Usuário em débito ou limite atingido.
VERIFICA DISPONIBILIDADE.	Pré-Cond. Pós-Cond.	
VERIFICA STATUS DA PUBLICAÇÃO	Pré-Cond. Pós-Cond.	
EFETIVA EMPRÉSTIMO	Pré-Cond. Pós-Cond.	Disponibilidade da publicação para o usuário. Não disponibilidade da publicação se for última cópia e total de empréstimo do usuário atualizado.
FAZ NOVA RESERVA	Pré-Cond. Pós-Cond.	Não disponibilidade da publicação. Ter sido efetuada a reserva de uma publicação.

UNIDADE DE INFORMAÇÃO	RESTRIÇÕES
Pub. Ausente	Não é uma Publicação Disponível
Pub. Disponível	Não é uma Publicação Ausente
Pub. Disponível c/Reserva	Não é uma Publicação Ausente

Fig. 5: Lista de Condições

que ser verificado se pode ser usada, uma das relações "default" (compare com a seção 3.2). O passo 2 pode ser dividido nos seguintes subpassos:

a) Para cada unidade de informação do GC que foi introduzida durante o último ciclo de iteração, deve ser examinado se pode ser representada por novas relações entre as classes de entidade existentes ou pela introdução de uma nova classe.

b) O GC é analisado posteriormente para identificar relações adicionais que são requeridas no GE para expressar restrições que devem ser mantidas entre as entidades (classes de entidades). Esta análise é orientada inicialmente para relações que podem ser derivadas das entidades de unidades de informação de entrada/saída de uma função.

c) O GE é examinado em relação a diversos critérios de modelagem que serão estabelecidos pelo projetista do GC.

   Exemplos de tais critérios:

   - As duas relações 'C1 é parte de C2' e 'C1 é elemento C2' não devem coexistir entre C1 e C2.
   - Se é definida uma relação 'C1 é um C2' com cardinalidade (1,1) então não é permitida uma segunda relação 'C2 é um C1' com cardinalidade (1,1).
   - Se existem duas relações 'C1 é um C2' com cardinalidade (1,1) e 'C2 é um C1' com cardinalidade (0,1), então de uma relação 'C2πC3' pode ser deduzida uma relação 'C1πC3' que portanto não precisa ser incluída no GE (veja figura 3).

Após a definição do GE obtém-se uma primeira visão do SI.

Passo 3:

A tabela de acessos é construída de modo a especificar para cada classe de entidade e para cada relação do GE a maneira de sua manipulação pelas funções do GC. Esta análise é requerida pois os arcos do GC não especificam se as entidades de uma unidade de informação são lidas, consumidas, modificadas ou geradas. Entretanto, esta espécie de informação é essencial para a posterior derivação da rede THM.

A tabela de acessos é organizada como uma matriz com as linhas representando as classes de entidades e as relações e com as colunas representando as funções. Um elemento da matriz é preenchido com G/C/L/M se uma função correspondentemente gera/consome/ler/modifica a classe de entidade ou relação. (Veja figura 4). Se uma classe ou uma relação não tem uma entrada na tabela de acessos especificando a espécie de acesso para tal, é requerido um novo ciclo de iteração paraintroduzir as funções e eventos correspondentes.

Passo 4: Análise das Funções

Para determinar se uma função está definida exatamente, deve ser verificado se:

- Existem situações em que nem todas as unidades de informação de entrada/saída são acessadas pela função,

- as pré/pós-condições especificadas para a função cobrem alternativas que devem ser representadas por diversas subfunções.

Tendo identificado tais situações, cada uma das transições de estado alternativas que são representadas por uma função, são representadas por uma subfunção correspondente. Geralmente, durante este processo de decomposição, as regras para morfismos de redes não devem ser violadas (Veja |4|).

O exemplo de automação do controle do acêrvo bibliográfico de uma biblioteca:

Ao longo do texto foi apresentado, sem maiores detalhes, o sistema de controle do acêrvo bibliográfico de uma pequena biblioteca. Na figura 2a apresentamos uma visão global do sistema. As funções "Catalogação de Documentos" e "Empréstimo" foram decompostas em um GC de segundo nível (figura 2b e 2c, respectivamente). A figura 2d é uma decomposição (detalhamento) da função "verifica disponibilidade" que foi apresentada no segundo nível (figura 2c) sendo, portanto, um GC de terceiro nível. "Documento", "Resposta-a-consulta" e "Relatório de publicações destruídas" são exemplos de unidades externas de informação. "Publicação" e "Usuário" são exemplos de unidades internas de informação. "Chegada de documento" e "Chegada de ordem de destruição" são exemplos de eventos (externos) de ativação de uma função e a ocorrência de qualquer dos dois irá ativar a função "Registro".

Na figura 5 são encontradas as pré/pós-condições e restrições descritas na seção 3.3. A figura 3 mostra o Grafo Estrutural correspondente ao GC da figura 2.

Conclusão

Foi apresentado um caso prático de desenvolvimento do esquema conceitual para um sistema de informações usando uma nova metodologia de projeto. A metodologia usada é bastante promissora por estabelecer uma boa base para comunicação entre o especialista de aplicações e o ABD já que o Modelo de Requisitos desenvolvido na primeira fase é orientado para o especialista de aplicações enquanto que a rede THM a ser desenvolvida na segunda fase é orientada para conceitos de banco de dados.

Referências

|1| Brodie, M.L.: Association: A Data Base Abstraction for Semantic Modeling, em Chen(ed): Entity-Relationship Aproach to Information Modeling and Analysis, 1981.

|2| Bracchi, C.; Pernici, B.: The Design Requirements of Oficce Systems, em: ACM Trans. on Oficce Information Systems 2,2 (Abril 1984), 151-170.

|3| Genrich H.J.; Lautenbach, K.: System Modelling with High-Level Petri Nets, em: Theoretical Computer Science 13(1981), 109-136.

|4| Genrich H.J.; Stankiewicz-Wiechno, E.: A dictionary of Some Basic Notations in Net Theory, em: Brauer(ed.): Net Theory and Application, LNCS 84, Springer Verlag, 1980.

|5| Gustafsson, M.R. et al.: A Declarative Aproach to Conceptual Information Modelling, em: Olle et al(eds.): Information Systems Design Methodologies: A Comparative Review, North-Holland Publ. Co., 1982.

|6| Hammer, M.; McLeod, D.: Data Base Description with SDM: A Semantic Data Base Model, em ACM TODS 6,3 (1981).

|7| Horndasch, A.; Studer, R.; Yasdi, R.: An Approach to Conceptual Schema Design of Information Systems, em Proc. IFIP WG 8.1 Working Conference on Theoretical and Formal Aspects on Information Systems, A. Sernadas e A. Olivẽ (eds.) Sitges, North-Holland Publ. Co., 1985.

|8| Lundeberg, M.: The ISAC Approach to Specification of Information Systems and its Application to the Organization on an IFIF Working Conference, em: Olle et al (eds.): Information Systems Design Methodologies: A Comparative Review, North-Holland Publ. Co., 1982.

|9| Olle, T.W. et (eds.): Information Systems Design Methodologies: A Comparative Review, North-Holland Publ. Co., 1982.

|10| Richter, G.: IML - Inscribed Nets for Modelling Text Processing and Data (Base) Management Systems, em Proc. 7th Int. Conf. on Very Large Data Bases, Cannes, 1981.

|11| Richter, G.: Utilization of Data Access and Manipulation in Conceptual Schema Definitions, em: Information Systems 6, (1981), 53-71.

|12| Richter, G.; Durchholz, R.: IML - Inscribed High-Level Petri Nets, em: Methodologies: A Comparative Review, North-Holland Publ. Co., 1982.

|13| Ross, D.T.: Structured Analysis: A Language for Communicating Ideas, em: IEEE Trans. on Software Engineering 3,1 (1977).

|14| Schiel, U.: An Abstract Introduction to the Temporal-Hierarchic Data Model (THM), em: Proc. 9th Int. Conf. on Very Large Data Bases, Florence, 1983.

|15| Schiel, U.: A Semantic Data Model for Conceptual Schemas and their Mapping to Internal Relational Schemas (en Alemão), University of Stuttgart, Doctoral Thesis, 1984.

COMPUTARIZACION DE INFORMACION BIBLIOGRAFICA. APLICACION DE TEORIA DE AUTOMATAS Y DE CONSULTAS INTERACTIVAS.

S. TENZER, R. LAVEGA

RESUMEN

Se implementó un sistema computarizado de apoyo para el almacenamiento, la clasificación y la recuperación, total o selectiva, de información bibliográfica. Los procesos interactivos son en tiempo real y operados por bibliotecarias. La consistencia de información se lleva a cabo mediante un autómata finito determinístico. Las consultas interactivas se soportan a través de cuatro estructuras de datos indexadas eficientes.

ABSTRACT

A computerized support system for storing, sorting and retrieving bibliographic information has been implemented. The librarians perform by themselves the on-line interactive data entry and consultations. The Information checking is performed through a Finite Deterministic Automaton. The interactive retrieving is based on four eficient indexed data structures.

INTRODUCCION

El Centro Latinoamericano de Perinatología y Desarrollo Humano - CLAP, desde hace quince años, trabaja en investigación, docencia y cooperación técnica con los países de la Región de las Américas, en el campo de la perinatología y el desarrollo humano. Para lograr sus objetivos, el Centro posee una biblioteca especializada, en la cual el manejo manual de la información bibliográfica era el problema principal. En particular, no había índices disponibles sobre las propias publicaciones del CLAP, que superan el millar.

Debido al grado de especialización de la Biblioteca los índices internacionales, tales como por ejemplo los confeccionados por BIREME - Biblioteca Regional de Medicina y Ciencias de la Salud (San Pablo) no satisfacen las necesidades del Centro.

Desde 1980, el CLAP tiene su propia computadora, una PDP11/34, para todos sus propósitos: manejo de los bancos de datos perinatales, procesamiento de información epidemiológica, investigación, docencia de estadística e informática aplicada a la salud, etc. Se decidió hacer uso de la computadora para el tratamiento de datos no-numéricos a efectos de contar con un sistema para clasificar y recuperar la información bibliográfica.

IMPLEMENTACION

El Sistema se implementó a lo largo de varios años, agregándose sucesivamente nuevas facilidades.

Para definir la implementación fue necesario analizar, estudiar y documentar todo el trabajo de la Biblioteca, con énfasis en la selección de entradas bibliográficas y la indización de los documentos, que incluyen: publicaciones periódicas, apartados, informes, tesis, libros, congresos y repartidos. Sólo en apartados hay más de 20,000.

Se diseñó el sistema y se estableció que la transcripción a medio magnético de las entradas bibliográficas fuera mediante reglas muy similares a las empleadas en bibliotecología.

Las entradas bibliográficas se llaman asientos. Cada asiento contiene los datos requeridos para identificar y clasificar un documento. Ellos son:

- Código de publicación
- Autor y coautores
- Título
- Publicación (pie de imprenta y colación)
- Notas:

- Palabras claves (descriptores)
- Referencias
- Resumen
- Otros

- Forma en que se encuentra en la Biblioteca

El código de publicación identifica al tipo de documento clasificado, el cual puede ser: libro, parte de un libro, congreso, parte de un congreso, tesis, artículo de publicación periódica, informe o repartido.

Los programas fueron codificados en lenguaje COBOL, específicamente COBOL Plus (con CLASS para el diseño de las pantallas) bajo sistema operativo TSX. Actualmente el Sistema se compone de 79 programas.

Las pantallas para tareas interactivas se han diseñado en forma sencilla y cuidando la estética. Además, el pedido de campos asegura la especificación válida de parámetros.

El Sistema cuenta con módulos interactivos a cargo de las propias bibliotecarias y otros en tiempo diferido, a cargo de operador de computadora.

Los módulos en línea están guiados por menúes que aparecen en la pantalla de la terminal de video, permitiendo que las bibliotecarias realicen su tarea de manera cómoda y segura.

MODULOS

Los módulos del Sistema son los siguientes:

Propósito:	Modalidad:	A cargo de:
Ingreso y corrección de asientos	ON LINE	Bibliotecaria
Consistencia de asientos y otros controles	BATCH	Operador
Emisión de índices impresos	BATCH	Operador
Definición del diccionario de descriptores	ON LINE	Bibliotecaria
Generación de estructuras para consulta interactiva	BATCH	Operador
Consulta interactiva	ON LINE	Bibliotecaria
Estadísticas	BATCH	Operador
Tareas complementarias	ON LINE	Bibliotecaria

En el módulo de ingreso y corrección de asientos, las bibliotecarias, se encargan de la preparación de los datos. Para ello cuentan con un menú en pantalla, un programa editor de textos y el manual de reglas de transcripción.

El módulo de consistencia de asientos y otros controles es operado periodicamente. Se realizan los controles de los asientos ingresados y corre-

gidos, y se incrementa el banco de datos con los correctos. Los controles son:

- sintaxis de los asientos (aplicando técnicas de teoría de autómatas)
- asientos con número de inventario repetido
- detección de posibles asientos duplicados (por semejanza en texto)
- descriptores no definidos.

Los errores son corregidos por las propias bibliotecarias.

El módulo de emisión impresa de índices clasifica todos los asientos según el concepto de interés y presenta la información en un único formato tipo. Ver figura 1.

Se emiten tantas copias como se deseen o bien un original, que sirve para la impresión a mimeógrafo o en offsett. Los índices disponibles son los ordenados por: Nro. de inventario, Título, Autor, o Decreciente por fecha y autor. También se especifica la inclusión, o no, de los resumenes.

CLAP
INFORME DEL CLAP 1970-1975.INVESTIGACIONES REALIZADAS.
OPS, REUNION DEL COMITE EJECUTIVO DEL CONSEJO DIRECTIVO, WASHINGTON, JUNIO, 1976
37P.
APARTADO NRO.INV. 1067700 I - INFORME

CLAP:
NARRATIVE REPORT OF THE ACTIVITIES ACCOMPLISHED BY THE CLAP.
OPS, WASHINGTON, , 1973
APARTADO NRO.INV. 1057900 I - INFORME

CLAP
INFORME DE LAS ACTIVIDADES REALIZADAS EN EL PRIMER SEMESTRE DE 1979.
OPS, WASHINGTON, 1 EN. - 30 JUNIO., 1979 33P.
APARTADO NRO.INV. 1083300 I - INFORME

COCH, J.
DEVELOPMENT OF A NEW METHOD FOR THE ISOLATION OF OXYTOCIN FROM HUMAN BLOOD.
OPS
RESEARCH IN PROGRESS. 1974.
WASHINGTON, OPS, 1974, P.132-134
REF.
L*W20.5-PANR NRO.INV. 1059210 LP - LIBRO-PARTE

COLLIA, F.
DIAZ, A. G., FESCINA, R., HORCHER, R.,
RIMOLI, R., SCHWARCZ, R., TENZER, S.,
URMAN, J.
DIAGNOSTICO DE SITUACION PERINATAL. RESULTADOS OBTENIDOS CON LA APLICACION DE LA HISTORIA CLINICA PERINATAL EN EL HOSPITAL SARDA DE BUENOS AIRES.
MONTEVIDEO, CLAP, 1978, 39P.
APARTADO NRO.INV. 1078600 L - LIBRO O FOLLETO

CORVANO, M. C.
CRECIMIENTO FETAL INTRAUTERINO EN EMBARAZOS GEMELARES. PATRONES ECOGRAFICOS.
MONTEVIDEO,,URUGUAY CLAP, 1981, 97P.
TESIS DE DOCTORADO, REF., RES.
APARTADO NRO.INV. 1093500 T - TESIS
EL CONOCIMIENTO DEL CRECIMIENTO FETAL ES UNO DE LOS CONCEPTOS BASICOS QUE DEBE DOMINAR TODO PROFESIONAL QUE TENGA LA RESPONSABILIDAD DE VIGILAR UN EMBARAZO.
EL ESTUDIO DEL FETO COMIENZA CON EL ESTUDIO DE LOS PATRONES DE CRECIMIENTO. ESTOS DEBEN SERVIR COMO GUIA PARA LA DETECCION, DIAGNOSTICO Y TRATAMIENTO DE LOS TRASTORNOS DE LA VIDA INTRAUTERINA.
Y CUANDO NOS ENCONTRAMOS FRENTE A UN EMBARAZO DE ALTO RIESGO POR SU GRAN CONTRIBUCION A LA MORTALIDAD PERINATAL Y POR LA FRECUENTE MORBILIDAD FETAL QUE ENTRAÑA, COMO LO ES EL EMBARAZO GEMELAR, LO ANTERIORMENTE DICHO COBRA AUN MAYOR IMPORTANCIA.
EL PRESENTE ES UN ANALISIS PROSPECTIVO, QUE CONSISTE EN UN ESTUDIO DE TIPO LONGITUDINAL DE EMBARAZOS GEMELARES, REALIZADO EN LA UNIDAD DE INVESTIGACION Y DIAGNOSTICO ECOGRAFICO DEL CLAP, DE 1979 A LA FECHA.
FUERON SEGUIDOS SEMANALMENTE DEL COMIENZO AL TERMINO DE LA GESTACION, 32 PACIENTES SANAS, PORTADORAS DE EMBARAZO GEMELAR, EN LAS QUE SE MIDIERON PARAMETROS CRANEANOS, ABDOMINALES Y RELACIONES ENTRE LOS MISMOS.
SE ANALIZARON LOS DATOS OBTENIDOS DE TODA LA POBLACION Y DE DOS GRUPOS SELECCIONADOS SEGUN PESO AL NACIMIENTO, GEMELARES MAYORES Y MENORES.
FUERON COMPARADOS ESTOS DOS ULTIMOS GRUPOS ENTRE SI Y CON LOS PATRONES DE CRECIMIENTO OBTENIDOS DE EMBARAZOS UNICOS.
LOS RESULTADOS OBTENIDOS MOSTRARON UN CRECIMIENTO DE GEMELARES MAYORES, EN LA MAYOR PARTE DE LOS INDICADORES, ASIMILABLE A LOS RESULTADOS DE LOS UNICOS.
LOS GEMELARES MENORES SE COMPORTARON COMO RETARDOS EN EL CRECIMIENTO INTRAUTERINO.
A TRAVES DE LA MONITORIZACION INTRAUTERO DEL CRECIMIENTO FETAL MEDIANTE LA TECNICA ULTRASONOGRAFICA, QUE PERMITE LA REALIZACION DE MEDIDAS DE FIELES INDICADORES DE LA EVOLUCION FETAL, ES POSIBLE DETECTAR UN RETARDO EN EL CRECIMIENTO INTRAUTERINO EN EL PARTO EN UNO DE LOS FETOS GEMELARES.

Figura 1: ejemplo de salida impresa.

El módulo de definición del diccionario de descriptores consiste en un subsistema para asegurar el uso de descriptores válidos. Actualmente la lista incluye casi 500 palabras claves. El propósito es mantener la mínima cantidad posible. El diccionario contiene sinónimos y comentarios. Ha sido confeccionado y se mantiene coordinado con BIREME.

Figura 2:

C . L . A . P . 26/03/85

C O N S U L T A B I B L I O G R A F I C A I N T E R A C T I V A

Opciones de Proceso.

1 - Ayuda (instrucciones operativas)

2 - Selecciona por Fecha, Autor y Descriptores

3 - Selecciona por Texto en el Título

4 - Visualiza trabajos seleccionados

5 - Clasifica índice de trabajos seleccionados

6 - Imprime los trabajos seleccionados

9 - Fin

Ingrese número de Tarea: _

El módulo de consulta interactiva permite la selección y la visualización en pantalla o en forma impresa de los asientos. Ver figura 2.
El usuario ingresa los parámetros de la consulta desde su terminal y el Sistema devuelve todos los asientos que son respuesta.
Las consultas son por:
- descriptores (hasta tres)
- autor y coautores
- fecha (en intervalo desde y hasta).

Pueden ser simples, especificando sólo un item de consulta o bien combinadas, especificando varios items, en cuyo caso la selección es tipo "AND".

Un Asiento puede estar clasificado bajo una, varias o ninguna palabra clave.
Puede contener un autor principal (no obligatorio) y uno, varios o ningún coautor. A los efectos de la consulta, el autor y los coautores se consideran indistintamente autores del asiento.

Puede contener, o no, una fecha, que es la de edición.

Las opciones de visualización de los asientos seleccionados incluyen (ver figura 3):

- impresión o despliegue
- asiento completo o sólo algunos campos
- recorrido ascendente por número de inventario o retroceso o bien al azar por número de inventario o por ordinal de selección.

Figura 3:

```
C . L . A . P .                                                    26/03/85
-------------------------------------------------------------------------
  C O N S U L T A   B I B L I O G R A F I C A   I N T E R A C T I V A
-------------------------------------------------------------------------

VISUALIZA trabajos seleccionados.                  SELECCIONADOS:      2

Intervalo de Fechas: 1970  A  1979    Autor: SCHWARCZ
Descriptores: RUPTURA PREMATURA DE MEMBRANAS
              FETO

Número de Inventario : 1059500    Ordinal: 00001
  Autores:
          SCHWARCZ                          R.
          BELIZAN                           J. M.
          NIETO                             F.
          TENZER                            S.
P. claves:
          AMNIOTOMIA
          FETO
          PARTO
          RECIEN NACIDO
          RUPTURA PREMATURA DE MEMBRANAS
Otros elementos:
          LA ROTURA PRECOZ DE LAS MEMBRANAS OVULARES Y SUS EFECTOS SOBRE
          EL PARTO Y EL NEONATO.  INVESTIGACION COLABORATIVA ENTRE
          MATERNIDADES DE AMERICA LATINA.
            MONTEVIDEO,
            CLAP,
            1975,
            80P.
            RES.
           A
(A-Ant. S-Sigte D-Directo)-Ver  (R)-Reinicia  (9)-Finaliza ..... :
```

El módulo de estadísticas consiste en las salidas impresas que informan sobre la productividad en ingreso de asientos. También se generan estadísticas sobre la frecuencia de aparición de los nombres de autores y coautores. En forma similar, se obtienen estadísticas sobre las palabras que componen los títulos de los documentos. Esta información es útil en la especificación de selección de asientos por texto contenido en el título.

El módulo de tareas complementarias se compone de subproductos obtenidos por las bibliotecarias, prácticamente sin intervención de técnicos en computación. Consiste en el uso de las facilidades existentes en el Sistema, como ser el editor de textos, para confeccionar cartas a los morosos, preparar bibliografías especiales, etc. Este es un resultado secundario importante a destacar por la iniciativa y autosuficiencia de las bibliotecarias.

AUTOMATA FINITO DETERMINISTICO

El control de consistencia de los asientos bibliográficos es realizado por una máquina de Turing combinada con un Autómata Finito Determinístico cuya fórmula es:

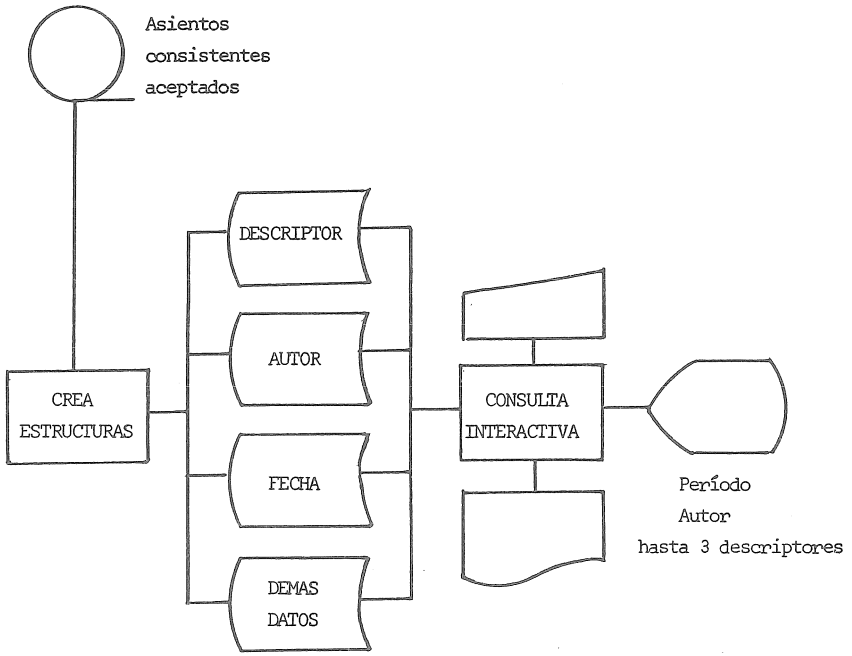
$$M=(I, O, Q, f, g, q0, F)$$

M: máquina secuencial que acepta los asientos y genera una salida.
I: conjunto de los 64 caracteres de entrada (letras mayúsculas, números y caracteres especiales).
O: conjunto de 70 símbolos de salida.
Q: conjunto de posibles estados (son 80).
f: QxI -- Q función de estado siguiente.
g: QxI -- O función de salida en versión Mealy.
q0: estado inicial.
F: conjunto de posibles estados finales.

Las funciones "f" y "g" se almacenan en forma de matriz de transición, la cual puede ser alterada fácilmente, para incluir otros documentos o modificar las reglas de transcripción.

Debido a que la PDP11 con TSX permite time sharing con particiones de hasta 59 KB fue necesario dividir el programa en dos asincrónicos intercomunicados. Uno de ellos es la matriz y el otro el programa propiamente dicho.

Figura 4: Esquema de consulta interactiva



BASE DE DATOS PARA CONSULTA INTERACTIVA

Para permitir la consulta interactiva fue necesario crear estructuras de datos que soporten las relaciones de búsqueda y que sean eficientes en tiempo de respuesta y memoria.

A partir del archivo secuencial de asientos consistentes aceptados se generan cuatro estructuras de datos, indexadas (ver figura 4) que son:

- descriptores (palabras claves)
- autor
-fecha
- demás datos del asiento

El archivo de demás datos del asiento está compuesto sólo por aquellos elementos que no se encuentran en las estructuras anteriores.
Se trata de introducir redundancia controlada, sólo la imprescindible, para ahorrar espacio en disco.

Cada asiento se identifica con un número de inventario único, que es el campo de vínculo entre todas las estructuras.

Las tres primeras estructuras, que son las de búsqueda, son lógicamente análogas y sólo difieren en contenido. Por lo tanto, a modo de ejemplo se sintetiza la correspondiente a descriptores.

ESTRUCTURA DE DESCRIPTORES

Un Asiento puede estar asociado a uno, varios o ningún descriptor. Un descriptor puede estar asociado a uno o varios asientos. La relación: Número de inventario - Descriptor no es biunívoca.
Se define así: Número de inventario -≫Descriptor
Descriptor ----------≫Número de inventario

Permite accesos al azar y secuencial lógico por Descriptor y por Número de inventario.

Cada descriptor puede tener hasta 35 caracteres. Para reducir el factor de ocupación del disco se define como clave los primeros "n" caracteres, en este caso 15. El acceso es por un prefijo y por tanto se implementa una rutina de búsqueda con sinónimos.

En la estructura se mantienen cuentas de cantidad de asientos por descriptor. Entonces, para realizar la búsqueda el Sistema primero determina cual es la cadena (descriptor) más corta (que tiene menos asientos) para minimizar los accesos.

Ejemplo de consulta simple: el usuario selecciona por un sólo descriptor, todos los asientos clasificados bajo 'RECIEN NACIDO'. Se accede al azar por palabra clave y se recorre la cadena secuencial lógico. Todos los asientos de la cadena serán respuesta a la consulta.

Ejemplo de consulta combinada: el usuario selecciona por tres descriptores, todos los asientos clasificados bajo 'PESO AL NACER' y 'RECIEN NACI-

DO' y 'ALCOHOLISMO'. Se accede al azar por una Palabra Clave (la que tenga Cuenta menor) y se recorre esta cadena. Para cada elemento de la cadena se obtiene el número de inventario asociado, y se accede directo a la misma estructura por número de inventario investigando que las demás palabras claves de este Asiento cumplan con la condición. Si es así entonces el Asiento es respuesta a la consulta.

Con todos los asientos que son respuesta a la consulta se genera un archivo llamado Indice Respuesta. Contiene una entrada por cada Asiento que es respuesta, que básicamente es el número de inventario.
Se define así el vínculo para el acceso a todos los elementos que componen un asiento a los efectos de la posterior visualización.

CONCLUSION

El uso de la computadora en bibliotecología reduce el trabajo del personal, mejora la calidad de los datos, facilita la indización del material y su recuperación combinada, brindando servicios más adecuados a los usuarios.

BIBLIOGRAFIA

1) American Library Association. Reglas de Catalogación Angloamericanas. Washington, Organización de los Estados Americanos, 1970. (Manuales del Bibliotecario No.7)

2) Becker, J., Hayes, R.M. Information storage and retrieval: tools, elements theories. Somerset, New Jersey, Wiley, 1967.

3) Booth, T.L. Sequential machines and automata theory. Somerset, New Jersey, Wiley, 1967.

4) Jordan, J.R. Let the computer select your reading list. Datamation, 16(2):91-94, 1970.

5) Knuth, D.E. The art of computer programming. v.1. Addison Wesley, 1972.

6) Maegerlein, R.C. A simple-minded documentation system. Datamation, 16(2):131-132, 1970.

7) Matthews, F.W., Shillingford, A.D. Variations of KWIC. Aslib Proceedings, 25(4):140-152, 1973.

8) Salton, G. Automation information organization and retrieval. New York, Mc. Graw-Hill, 1968.

9) Starmer, F.C., Rosati, R.A., Simon, S.B. Interactive acquisition and analysis of discrete data. Computers and Biomedical Research, 5(5):505-514, 1972.

10) Tenzer, S.M. Análisis del sistema de información de la Biblioteca del Centro Latinoamericano de Perinatología y Desarrollo Humano. Montevideo, CLAP, 1973. (Publicación científica CLAP No. 574,01)

11) Tenzer, S.M. Automatización de la Biblioteca. Instructivo de transcripción de los asientos bibliográficos a tarjetas perforadas, Montevideo, CLAP, 1974. (Publicación científica CLAP 574,02)

12) Tenzer, S.M. Automatización de la Biblioteca, Reglas de transcripción de los asientos bibliográficos a computadora, Montevideo, CLAP, 1984. (Documento Interno CLAP 6/84)

13) Tenzer, S.M., Lavega, R., Uzal, M. Sistema de automatización de la Biblioteca. Sub-sistema ingreso de datos, Instrucciones de uso de la terminal de computadora. Montevideo, CLAP, 1983. (Documento interno CLAP 34/83)

14) Yerke, T.B., Computer support of the researcher's own documentation. Datamation, 16(2): 75-77, 1970.

R. GALLARD (*)

RESUMEN

El presente trabajo propone una extensión en la definición de la ejecución de Redes de Petri a fin de eliminar las limitaciones de estas redes en la modelización de sistemas reales, cuando uno desea respuestas a preguntas que van mas alla de un "si" o un "no" al analizar la aceptabilidad del comportamiento del sistema modelizado.

El mayor contenido de información en las respuestas del modelo se logra permitiendo al usuario entrar información acerca del conocimiento que el mismo tiene sobre algunas características de los requerimientos del sistema.

En consecuencia sera necesaria la introducción de un vector de probabilidades de ejecución de eventos habilitados, como una componente extra en cada nodo del arbol de alcanzabilidad y el uso de técnicas de Montecarlo.

ABSTRACT

In the present work an extension in the definition of a Petri Net execution is given in order to eliminate the limitations of these nets in the modelling of real systems, when one wishes to answer to questions beyond a "yes" or a "no", in analysing the acceptable behaviour of a modelled system.

The greater content of information in the answers of the model is achieved by allowing the user to enter information about the knowledge he has on some characteristics of system requirements, before the execution of the net.

Therefore, the introduction of an Execution Probability Vector for enabled events, as an extra component in each node of the Reachability Tree and the aid of Montecarlo techniques, are necessary.

(*) Computador Científico (UNBA 1975), MSc in Computer Science (University of Aston in Birmingham, U.K. 1982), Profesor Titular en la Escuela de Informatica de la Universidad Nacional de San Luis, Cátedras de Sistemas Operativos y Analisis Comparativo de Sistemas.
Chacabuco y Pedernera , 5700 - San Luis Argentina.

This paper is a consequence of the work of the author and coworkers in the study and implementation of Software for the sintax control, execution and analysis of Petri Nets.

In previous works it became clear for the group, as Peterson and other authors stated, the ability of these nets to detect pathological behaviours (deadlocks, violations of mutual exclusion restrictions, etc) in systems that exhibit concurrent activities.

For this purpose a Petri Net is a simple and powerful tool which can serve as a basis for ambitious projects, as that of Nelson, Haibt and Sheridan; "Specification, Design and Implementation via Annotated Petri Nets" [4]. In general, we notice that a Petri Net is a highly efficient medium for either the diagnosis of systems susceptible of exhibiting this type of failures or the elimination of these failures during the debugging stages.

Nevertheless, and this is a limitation in the definition of the execution of a Petri Net, this does not occur when one wishes to more faithfully model the behaviour of a real system.

Such a limitation is a consequence of the basic assumption that says that, any transition in a set of enabled transitions can fire. That "any" is translated, in the software which implements the execution, as a random selection of the transition to be fired with a uniform probability distribution (equiprobable).

This type of implementation is reasonable in the cases where one wishes to show,by any means (e.g the construction of the reachability tree or the execution of a graph), the reachability of an abnormal state in the system.

But, it is clear that for the one who wants to obtain more information than the mere detection of failures, such type of implementation is unsatisfactory.

For example, if a set of enabling tokens in some places (conditions which describes the state of the system) determines that certain transitions (possible events in the system) can fire, it is not always true that all the enabled events are equally probable to occur in the real system.

Otherwise, it is true that in the presence of many requeriments of a scarce resource from many processes there would be a priority policy to grant them. It is also true that the access to a particular type of resource could be more frequent than the access to other type of resources.

Such pre-stablished preferences in the real system are not possible to model with the simple assumption that any transition of an enabled set can fire without adding anything else.

The present work has the purpose of defining a selection function, to select a transition of a set of enabled transitions, based on the knowledge the designer or analyst has about the system and of explaining briefly, some details of a possible implementation.

PREVIOUS CONCEPTS

Given a marked Petri Net M = <S,μ>

where
μ is an initial marking and S is the Petri Net Structure defined by:

S = <P,T,I,O>

with P = {p1,p2,...,pn} Set of n places, n >= 0
T = {t1,t1,...,tm} Set of m transitions, m >= 0
I : T --> P' Input function
O : T --> P' Output function
P' Set of all possible bags formed with elements of P,

the state space R(S,μ) of the network is formed by the markings which are reachable from the initial marking μ.

This Reachability Set, R(s,μ), has a finite representation by means of a Reachability Tree. Each node in this tree represents either a reachable marking or a set of reachable markings if the Petri Net is unbounded (presence of ω symbol in the tree).

In addition, for each of these nodes the subset Tau of enabled transitions is uniquely determined. This is true even in the cases where the ω symbol is present in a particular node, because the presence of extra tokens in a place affects neither the enabling nor the firing of transitions.

Consequently there is a unique set Tau of transitions that can fire for a particular marking (state of the Petri Net).

After highlighting these concepts we go now to the construction of the selection function E.

CONSTRUCTION OF THE SELECTION FUNCTION "E"

We describe now the building-up process for the selection function.

i)
For a given marking μ' ∈ R(S,μ), we define as an Enabling Vector a m-vector H = (h1,h2,...hm) such that

$$h_i = \begin{cases} 1 & \text{if } t_i \text{ is enabled in } \mu' \\ 0 & \text{otherwise} \end{cases} \qquad 1 \leq i \leq m$$

ii)

We associate to each enabling vector a weighting vector W which indicates the weighting that one knows is made in the real system, about the possibility of execution for each of the competitive and enabled events in the currently modelled state of the net (described by the enabled transitions).

Example :

> If for a given marking we have M = (1,0,0,1) and the associated weighting vector is W = (3,0,0,1), we would be describing a situation where, being enabled transitions t1 and t4, t1 has a probability 3 times greater than t4 to be fired.

This W vector is subsequently "normalized" to W' where

$$W' = W / \sum_i wi = (3/4, 0, 0, 1/4)$$

Briefly, W' defines a probability distribution for the sthocastic variable "transition to be fired".
Each component wi' of W' describes the firing probability of transition i

$$wi' = Pr \ (\text{ firing ti }).$$

This distribution is a probability one because

$$0 <= wi' <= 1 \qquad \text{and} \qquad \sum_i wi' = 1$$

iii)

Once the vector W' is obtained and by means of the well known application of the Montecarlo method, we have a decision rule to determine the transition to be fired.

If we call TAU to the class of all sets (Tau) of enabled transitions to be generated during Petri Net execution, we see that our selection function, so constructed, takes a Tau and for this Tau allows us, based in the knowledge one has about the system, to define a vector W' which is a descriptor of a probability distribution f to which applying the Montecarlo techniques returns us the transition to be fired.

The last paragraph can be rewritten as

$$E : TAU \longrightarrow Tau$$

that can be decomposed in

$$E1 : TAU \xrightarrow{m} M$$
$$E2 : M \xrightarrow{m} N$$
$$E3 : N \xrightarrow{m} Q$$
$$E4 : Q \xrightarrow{m} F$$
$$E5 : F \longrightarrow Tau$$

$$\therefore \quad E = E5 \ . \ E4 \ . \ E3 \ . \ E2 \ . \ E1$$

TAU : The class of all sets of enabled transitions for a given Petri Net, where each set corresponds to a particular reachable marking.

H^m : The set of all m-vectors whose elements are binary digits.

N^m : The set of all m-vectors whose elements are positive or null integers.

Q^m : The set of all m-vectors whose elements are positive rationals, less than or equal to 1 and they add up to one.

F : The set of all discret probability distribution functions with m elements in their domain.

Tau : The set of enabled transitions for a given reachable markings in the Petri Net.

SOME SUGGESTIONS FOR THE IMPLEMENTATION

i)

The concept of a node in the reachability tree, will be extended for a routine of tree generation, in order that each of these nodes will be formed by the pair:

$$(\mu,H) = (\text{ Current Markings, Current Enablins Vector })$$

ii)

Afterwards it will be necessary to write a routine (*) that, in traversing the tree and based on the user's knowledge, creates the corresponding weighting vector W for that particular state in the net. From this, the probability vector W' can be determined and then the enablins vector can be replaced in the node, which now takes this form

$$\text{Tree node} = (\mu,W')$$
$$= (\text{ Current Markings, Probability Vector associated with this markings })$$

iii)

Durins Petri Net execution, in each one of its states, the vector W' associated with the current marking will be fetched and a routine to apply the Montecarlo method will be called. In this way the transition to be fired is determined.

(*) It could be the same decribed in i).

CONCLUSIONS

As stated in the introduction, this type of implementation in the execution of a Petri Net would allow a behaviour of the model, granularly closer to that of the system state modelled in each instance of the net.

In the cases in which the tokens' distribution in the net is modelling the requirements of certain resources, it is possible for us to observe the accumulation of tokens in some places, and this fact would be modelling the feeding of requirements queues, whose parameters (mean and maximum queue lengths, mean waiting time in the queue measured in number of firings, etc) could be registered for further analysis.

The assignation of a firing probability equal to 1 or 0 to one of the enabled transitions, would naturally describe a priority policy without needing the use of inhibitor arcs.

As regards to the analysis of Liveness and Coverability, related with the problem of reachability, we see now that in the case of being able to answer affirmatively to questions such as :

- Is μ' reachable from μ ?
- Is there a reachable marking μ' from μ, such that μ' covers a given marking μ''?
- Is it possible for the net to arrive to state where no transition can fire ?
- Is there any reachable marking where the mutual exclusion restrictions are violated ?

we are now able not only to answer "yes", but also we would further precise our answer specifiying the arrival probability to such a state after k firings.

This is possible because,

$$\text{if } \mu' = \hat{\delta}(\mu, \text{Sigma})$$

where $\hat{\delta}$ is the extended next state function and Sigma is a firing sequence of length k ,then the arrival probability to μ' would be given by the product of the probabilities associated with the firing transitions in each of the succesive markings (states) through which the Petri Net goes.

Finally, we believe that this extended definition of a Petri Net execution improves its modelling ability allowing a more faithful representation of real systems behaviour and a better understanding of them by answering an important set of questions to which the traditional approach has no answer.

ACKNOWLEDGEMENTS

This work would not have been possible without the motivation supplied by Dr Alan J. Harget of the University of Aston in Birmingham (U.K.) who in 1982 introduced me in the subject.

I am also particularly grateful to the students Susana G. Barbenza and Mirtha S. Escudero for their colaboration in the development of a Petri Net Software and in general to the project group in Operating Systems of the Universidad Nacional de San Luis for their work in the area.

San Luis, Argentina Septiembre de 1984

BIBLIOGRAPHY

[1] C. Petri, "Kommunikation mit Automaten", Ph. D. dissertation, University of Bonn, Bonn, West Germany.

[2] J. Peterson, "Petri Nets", Computer Surveys, Volume 9, Number 3, September 1977.

[3] J. Peterson, "Petri Net Theory and the Modeling of Systems", Prentice Hall, 1981

[4] R. A. Nelson, L.M. Haibt and P.B. Sheridan, "Specification Design and Implementation via Annotated Petri Nets", IBM T. J. Watson Research Centre Yorktown Heights RC 9317 March 1982.

[5] R. Gallard, "A Set of Petri Net Routines for System Modelling" University of Aston in Birmingham, Birmingham, U.K.,July 1982.

C. M. DA COSTA*, S. S. TOSCANI**, T.B. CORSO***

SUMARIO

Este trabalho descreve a implementação do sistema operacional SONIX, compatível com o UNIX a nível de interface do usuário. O sistema foi escrito em Pascal Concorrente e um protótipo do mesmo encontra-se em funcionamento no minicomputador LABO-8034 do Curso de Pós-Graduação em Ciência da Computação da UFRGS.

ABSTRACT

This work describes the SONIX operating system which as UNIX compatible at the user interface level. The system was written in Concurrent Pascal and a prototype of it is already working on the LABO-8034 minicomputer of the Graduate Course in Computer Science at UFRGS.

* B.SC. Análise de Sistemas (PUC-RS, 1978), aluno de mestrado em Ciência da Computação (UFRGS), professor assistente do Departamento de Ciências Estatísticas e da Computação (UFSC); áreas de interesse: sistemas operacionais e linguagens de programação; Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99-Porto Alegre - RS - CEP 90.000.

** Engenheiro Eletricista (UFRGS, 1967), Mestre em Informática (PUC/RJ, 1969); áreas de interesse: programação concorrente e sistemas operacionais; professor Adjunto do Departamento de Informática/Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99 - Porto Alegre - RS - CEP 90.000.

*** Engenheiro Civil (UFRGS, 1973), Mestre em Ciência da Computação (UFRGS, 1978); áreas de interesse: programação concorrente e sistemas operacionais; Professor Adjunto do Departamento de Informática/Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99 - Porto Alegre - RS - CEP 90.000.

1. INTRODUÇÃO

Encontra-se em fase de conclusão, no Curso de Pós-Graduação em Ciência da Computação (CPGCC) da UFRGS, um projeto que tem por objetivo implementar a Máquina Pascal Concorrente (MPC), de forma microprogramada, no minicomputador ED-311.

No momento em que este projeto estiver concluído, todo o software disponível no sistema Pascal do LABO-8034 /MED 81/, incluindo-se os sistemas operacionais SOLO /HAN 77/ e DUO /CAR 81/, poderá ser transportado para o ED-311 sem problema algum.

Neste artigo é descrito um novo sistema operacional para a Máquina Pascal Concorrente que está sendo construido pelo grupo de pesquisa em Sistemas Operacionais do CPGCC/ /UFRGS. Este trabalho tem por objetivo oferecer melhores opções de uso para comunidade de usuários do CPGCC bem como desenvolver experiência na área de implementação de Sistemas Operacionais.

Na construção de um sistema operacional é necessário levar em conta dois aspectos muito importantes, quais sejam a especificação e a implementação do sistema especificado.

A decisão tomada foi o de implementar um sistema que já houvesse sido especificado anteriormente, de soluções inteligentes e que oferecesse aos usuários uma boa linguagem de comandos.

Um sistema com essas características e que, além disso, está se popularizando a ponto de muitos afirmarem tender a se tornar um padrão internacional é o UNIX, desenvolvido nos laboratórios da Bell, nos Estados Unidos /RIT 74/.

Assim, foi tomada a decisão de implementar um sistema em Pascal Concorrente, compatível com o UNIX a nível de interface do usuário, o qual foi chamado de SONIX.

2. DESCRIÇÃO DA IMPLEMENTAÇÃO DO NÚCLEO DO SISTEMA SONIX

A seguir será descrita a implementação do núcleo do Sonix. Seu detalhamento será feito em quatro partes. A primeira delas trata do controle de processos, a segunda da execução de programas sequenciais, a terceira da execução de procedimentos S-Shell e a quarta da inicialização do sistema.

2.1. Controle de Processos

No Sonix existem dois tipos de processos, processos Console e processos Envelope.

Os processos Console são os que interagem com o usuário; são criados quando da inicialização do sistema e executam o programa S-Shell que é o interpretador de comandos. Somente deixam de existir quando o sistema para de executar.

Processos Envelope, contrariamente, tem duração limitada. Um processo envelope é criado para executar um programa sequencial e "morre" após ter cumprido esta tarefa.

Como os processos Envelopes necessitam ser criados dinamicamente e na linguagem de implementação a criação de processos se dá de forma estática (o número de processos é definido em tempo de compilação), foi necessário criar um ambiente de simulação que permitisse tal coisa. A solução adotada foi empregar as primitivas de sincronização disponíveis na linguagem Pascal Concorrente, Delay e Continue, usadas nas estruturas monitor, para, juntamente com estruturas de dados e procedimentos, simular o ambiente necessário. Foi definido um monitor denominado Controlproc no qual foram implementados todos os procedimentos de sincronização e comunicação requeridos.

São mantidos no monitor Controlproc os registros descritores dos processos, bem como os procedimentos para criação e intercomunicação dos mesmos.

2.1.1 Criação de Processos

A criação de processos é feita da seguinte maneira: o processo pai executa o procedimento Fork do monitor Controlproc, o qual percorre os registros descritores de processos a procura de um que esteja no estado "D" (Desencarnado). Encontrado um nesta condição, seu estado é trocado para "V" (Vivo), sendo atualizados os demais campos do registro descritor. Após isto, o processo pai executa a operação Continue nesse processo, fazendo com que o processo filho passe a concorrer pelo uso do processador.

Registros Descritores de Processos

Cada processo é descrito no monitor Controlproc por um registro descritor, o qual é implementado pela estrutura de dados Desc que tem a seguinte constituição:

```
Type Estado = (V, D);
Type Desc   = Record
                    Número : Integer;
                    Atrib  : Integer;
                    Est    : Estado;
                    Idpai  : Integer
              End;
Var Descritor : Array (.0..Nproc.) of Desc;
```

Os campos deste registro possuem os seguintes significados:

Atrib : Contém a identificação do processo. E a mesma mantida no "kernel" da MPC, no registro descritor correspondente. Sua obtenção é feita pela instrução Attribute.

Est : Contém o estado do processo. Os valores possíveis são "V", significando que o processo esta "vivo", executando alguma ação, e "D" significando que o processo está "desencarnado", não estando executando nenhuma ação. Os processos Envelopes podem sofrer transição de um estado para outro. Os processos Console estão sempre no estado "V".

Idpai : Contém a identificação do processo pai.

2.1.2 Bloqueio e Sinalização de Processos

O procedimento padrão no Sonix é o processo pai criar um processo para executar um programa seqüencial e ficar a espera de que o processo filho termine a execução.

No momento em que isto ocorre, o processo pai é sinalizado pelo processo filho. Com isso o pai passa novamente a concorrer pelo uso do processador. Após ter sinalizado o processo pai, o processo filho "morre".

Primitiva Wait

O procedimento Waitm, do monitor Controlproc, faz com que o processo que o executa fique bloqueado a espera de sinalização. O bloqueio é feito quando o processo executa a instrução Delay no referido procedimento.

Primitiva Wakeup

O procedimento Wakeup, pertencente ao monitor Controlproc, faz com que o processo que o executa sinalize a seu processo pai, o qual passa a concorrer pelo processador. A sinalização é feita com a instrução Continue.

Primitiva Exit

O procedimento Exit, pertencente ao monitor Controlproc, faz com que o processo que o executa morra.

Isto é feito trocando o seu estado, no registro descritor, para "D". Após, isto o processo executa a operação Delay.

Primitiva Kill

O procedimento Kill, pertencente ao monitor Controlproc, permite que seja "morto" um determinado processo, especificado por seu número de criação. O estado do processo vítima é trocado para "D" e é executada a operação "Stop" no referido processo. Com isto, o programa seqüencial que o processo está executando será descontinuado na execução da próxima instrução virtual Falsejmp /MED 81/.

Todas as ações necessárias para "matar" o processo terão de ser efetuadas também sobre os seus descendentes, para que não sobrem processos "desgarrados" no sistema.

Nesta implementação da Kill o processo indicado sempre "morre", o que difere da implementação usual deste procedimento em outras versões Unix, nas quais o processo pode sobreviver /HOL 83/.

2.2 Execução de Programas Seqüenciais

Um programa seqüencial é executado transferindo seu código de disco magnético para a memória principal, colocando-o na área de dados do processo que irá gerenciar sua execução. Essa transferência é feita pela classe Loaderfile.

Existe também uma classe denominada Progstack que mantém as informações necessárias para permitir que programas seqüenciais possam chamar a outros programas sequenciais (chamadas aninhadas).

No Sonix existem três formas de execução para programas seqüenciais: através da criação de processos, sem envolver criação de processos e no modo "background". Estas três formas serão detalhadas a seguir.

2.2.1 Através da Criação de Processos

Esta é a forma básica usada pelo interpretador de comandos do sistema, o programa S-Shell. Quando o sistema é carregado, é criado um processo Console para cada terminal existente. O processo Console, por sua vez, carrega para sua execução o programa S-Shell, o qual fica recebendo comandos digitados pelo usuário. Sempre que o comando digitado implicar na execução de um programa seqüencial, será criado um processo para fazê-lo.

A criação do processo é realizada pelo procedimento Execpgm, o qual é invocado da seguinte forma:

Execpgm (Nameprog, Paramprog, Altera-IO, Recpipe);

Os parâmetros da chamada são definidos a seguir:

```
Type
    Identificador = Array (.1..20.) of Char;
    Regparam      = Array (.0..9.) of Identificador;
    Regaltera-IO  = Record
                        Ent, Sai : Boolean;
                        Nome-In, Nome-Out : Identificador
                    End;
    Pipetype = Record
                    Input, Output : Boolean;
                    Enderin, Enderout : Integer
               End;
Var
    Nameprog  : Identificador;
    Paramprog : Regparam;
    Alterá-IO : Regaltera-Io;
    Recpipe   : Pipetype;
```

Nameprog - Contém o nome do programa que deverá ser executado.

Paramprog - Informa os parâmetros que serão passados para o programa. Poderão ser no máximo dez.

Altera-IO - Contém informações sobre o redirecionamento de entrada e saída durante a execução do programa seqüencial. Os campos Ent e Sai indicam, respectivamente, alteração da entrada padrão e alteração da saída padrão. Nome-In e Nome-Out armazenam os nomes dos arquivos do redirecionamento.

Recpipe - Este parâmetro indica a criação de um canal de intercomunicação. Os campos Input e Output especificam para o processo se o mesmo vai depositar ou retirar dados do canal e Enderin e Enderout o endereço do canal através do qual os processos irão realizar a troca de mensagens.

O procedimento Excepgm cria um processo através da chamada do procedimento Fork do monitor Controlproc. O procedimento Fork devolve o número da entrada no array de registros descritores que descreve o processo criado. A seguir, Execpgm chama o procedimento Execpgm-M, do monitor Controlproc e passa todos os parâmetros recebidos quando da sua chamada e mais o número da entrada que descreve o processo criado.

O procedimento Execpgm-M armazena na estrutura de dados denominada. "Execute" todas as informações recebidas e em "Tabwork" a indicação de que o processo criado tem um programa seqüencial para executar. A estrutura Execute possui a seguinte definição:

```
Type Regexec = Record
                    Seqprog    : Identificador;
                    Paramprog  : Regparam;
                    Recpipe    : Pipetype;
                    Altera-IO  : Regaltera-IO
               end;
Var Execute : Array (.1..Nproc.) of Regexec;
```

O processo criado, após identificar que tem um programa seqüencial para executar, chamará o procedimento Fetchpgm, pertencente ao monitor controlproc, o qual simplesmente devolverá o conteúdo da estrutura de dados Execute. Após isso, o processo efetuará a chamada do procedimento Runpgm que, com o auxílio das classes Loaderfile e Progstack fará com que o programa seqüencial seja executado.

2.2.2 Sem Criação de Processos

Um programa qualquer em execução pode disparar um outro programa através da chamada do procedimento Execv.

Este procedimento tem a função de efetuar a chamada Runpgm passando as informações recebidas. A partir dai tudo se passa como descrito na seção anterior.

Como trata-se de uma chamada aninhada de programa, após o término do programa chamado, a execução deverá retornar ao programa que efetuou a chamada; para tanto o mesmo será recarregado e será restaurado seu contexto.

2.2.3 Modo "Background"

Basicamente, a função do S-Shell é a de ler, analisar e executar um comando. A leitura e a análise são realizadas pelo módulo Analisador enquanto a execução é realizada pelo módulo Executor. Quando a execução do comando implica na execução de um programa sequencial, o processo Console, que está executando o S-Shell, cria um processo filho para executar tal tarefa e espera pelo término do mesmo. Somente após isso volta a aceitar novos comandos do terminal.

Esta situação será alterada se o usuário disparar a execução de um comando em "background": o processo Console passará imediatamente a aceitar novos comandos do usuário, sem esperar pelo término da execução do comando anterior.

A ação de executar um comando em "background" pode ser dividida em duas fases. Na fase 1, quando o módulo Executor do programa S-Shell reconhece que o código virtual gerado pelo módulo Analisador deve ser executado em "background", chama ao procedimento Execbcg pertencente ao processo Console, passando como parâmetro o referido código. Esse procedimento cria um processo e faz a chamada Execbcg-M, pertencente ao monitor Controlproc, que tem a função de depositar na estrutura de dados "Background" o código gerado pelo programa S-Shell que deve ser interpretado pelo processo criado. Essa estrutura de dados possui a seguinte definição:

```
Const Lengthbcg = 20;
Type
    Tipooperando = (Inteiro, Caracter);
    Instrução    = Record
                            Flag: tipooperando;
                            Mnemonico : Integer;
                            Operando  : Identificador
                       End;
    Codetype = Array (.1..Lengthbcg.) of Instrução;
    Typebackground = Record
                          Code : Codetype;
                          Free : Boolean
                     End;
Var Background : Typebackground;
```

O campo Code armazena o código virtual a ser executado e o campo Free indica se a estrutura de dados está disponível ou não. Quando o código é armazenado Free recebe o va

lor False.

Imediatamente após o término do procedimento Execbcg o processo Console retorna ao contexto do programa S-Shell, estando apto, dessa forma, a aceitar novos comandos do terminal.

Na Fase2 o processo criado, ao identificador que deve executar um código em modo "background", chama o procedimento Fetchbcg, pertencente ao monitor Controlproc, que lhe devolve o código armazenado na estrutura de dados "Background" e torna essa estrutura disponível, (fazendo o campo Free receber o valor True).

Feito isso, dispara a execução do interpretador da linguagem de comandos (módulo Executor), o qual foi codificado como um procedimento dos processos. A partir desse momento é iniciada a interpretação do código virtual.

Sempre que for necessário, será criado um processo filho para a execução de um programa sequencial. Quando isso ocorrer, as coisas se passarão como descritas na primeira forma de execução de programas sequenciais que foi apresentada (através da criação de processos).

2.3 Procedimentos S-Shell

O processador da linguagem de comandos, denominado S-Shell foi programado como um programa seqüencial. Quando o sistema é inicializado, cada processo Console o carrega para a memória principal e começa a executá-lo.

O usuário, interagindo com o sistema através do S-Shell, poderá comandar a execução do programa S-Shell no modo "background" redirecionando sua entrada para um arquivo anteriormente criado e que contenha instruções S-Shell.

No referido arquivo poderão aparecer parâmetros posicionais referidos como $1,$2,..$n que serão substituídos pelos argumentos fornecidos pelo usuário ao disparar a execução de um Procedimento S-Shell, substituição esta que será feita antes de ser invocado o módulo Executor do processador da linguagem.

Como os procedimentos S-Shell são executados no modo "Background", o usuário continuará normalmente no terminal interagindo com a ocorrência (instanciada) do S-Shell carregada quando da inicialização do sistema.

2.4 Inicialização do Sistema

Quando o programa concorrente que implementa o sistema Sonix é carregado, sua execução começa pelo processo Ancestor (codificado como processo inicial na linguagem Pascal Concorrente), o qual efetua a operação "Init" no monitor Controlproc e, a seguir, nos processos do sistema (processos Sist), passando o referido monitor como parâmetro. A partir disso, todos os processos passam a concorrer pelo processador.

Cada processo Sist inicializa as classes Stackprog e Loaderprog e chama o procedimento Setatrib, pertencente ao monitor Controlproc. O procedimento Setatrib coloca no registro descritor do processo a sua identificação, a qual é obtida do "kernel" através da instrução Atributte. Ao término deste procedimento o processo "se mata" chamando o procedimento Exit do monitor em questão.

A partir disto, do ponto de vista do Sonix, somente existem o processo Ancestor e o monitor Controlproc. Na realidade existem "Nproc" processos do tipo "Sist" em delay no monitor Controlproc, esperando para serem criados" quando então desempenharão os papéis de processos Console ou processos Envelope.

No passo seguinte o processo Ancestor acessa o monitor Controlproc e configura o sistema, criando um processo console para cada terminal. O número de terminais é indicado pela constante Noterm, a qual pode ser alterada para refletir uma mudança no número de terminais da configuração.

Após ter realizado este trabalho, o processo Ancestor deixa de existir pois não tem mais nenhuma função a realizar.

Os processos Consoles criados carregam e começam a executar o programa S-Shell, passando dessa, forma, a tornar o sistema disponível aos usuários. Os processos envelopes serão "criados" pelos processos Consoles, durante a execução do S-Shell, para executarem comandos do usuário.

2.5 Estrutura dos Processos Sist

Um processo Sist pode tanto desempenhar o papel de um processo console como de um processo Envelope. O corpo principal dos processos Sist é o seguinte:

```
Begin
     Init Stackprog, Loaderprog;
     Estadoproc.Setatrib;
     Estadoproc.Exit;
     Cycle
          If Estadoproc.Myfunction = Console Then
          Begin
               Idok := False;
               While Not Idok do
                    Execident (Idok);
               With Execute do
               Begin
                    Seqprog := 'S-Shell'
                    Altera-io.Ent := False;
                    Altera-io.Sai := False;
```

```
                Recpipe.Input  := False;
                Recpipe.Output := False;
            End;
            Runpgm (Execute, Line, Result);
        End
        Else
        Begin
            Mywork := Indefined;
            While Mywork = Indefined Do
            Begin Estadoproc.Mywork;
                Wait;
            End;
            Case Mywork of
                Back : Begin
                         Estadoproc.Fetchbcg (Cod-back)
                         Interpretador (cod-back)
                       End;
            Procshell,
                Prog : Begin
                         Estadoproc.Fetchbcg (Executei)
                         Runpgm (Execute, Line, Result)
                       End
            End;
            Start;
            Estadoproc.Wakeup;
            Estadoproc.Exit;
        End;
    End;
End;
```

Quando o processo Ancestor configura o sistema, ele cria um processo Console (isto é, um processo Sist com a função de Console) para cada terminal existente.

Um processo Console ao receber o processador, identifica sua função e tenta reconhecer um usuário válido através do procedimento Execident.

Ao identificar um usuário no terminal, este processo prepara os parâmetros de forma adequada e dispara a execução do programa S-Shell através do procedimento Runpgm.

No momento em que o usuário encerrar sua sessão, terminando a execução do S-Shell, o processo console tentará a identificação de um novo usuário, repetindo, dessa forma, o procedimento descrito.

Os processos Console existirão enquanto o sistema Sonix estiver operando.

O sistema Sonix suporta "criação dinâmica" de processos para execução de programas seqüenciais, o que é requerido para implementar a linguagem de comandos disponível aos usuários.

Conforme já referido anteriormente, criar um processo significa chamar o procedimento Fork, do monitor Controlproc, o qual irá procurar nos registros descritores de processos pela existência de um no estado "D". Encontrando, colocará no registro descritor as informações adequadas e efetuará a operação Continue no processo correspondente, para que ele passe a executar a função de Envelope.

Como um processo Envelope é criado e começa a disputar o processador antes das estruturas de dados que informam o trabalho que o mesmo terá de realizar estarem atualizadas, ele fica em um laço chamando o procedimento Estadoproc.Myfunction, do monitor Controlproc, que lhe fornece esta informação. A instrução Wait, codificada neste laço faz com que o processo que a executa deixe de concorrer pelo processador por um pequeno espaço de tempo (um segundo); assim, tenta-se agilizar a atualização das estruturas de dados referidas.

O processo Envelope é capaz de realizar três tipos de trabalho. Se Mywork for igual a Back, o processo envelope executará um código em "background". Para isto, chamará o procedimento Estadoproc.Fetchbcg, pertencente ao monitor Controlproc, que lhe devolverá o código a ser executado. A execução desse código será feita através da chamada do módulo Executor. Se Mywork for igual a Procshell ou igual a Prog, o processo Envelope executará um programa seqüencial. Neste caso, chamará o procedimento Estadoproc.Fetchpgm, pertencente ao monitor Controlproc, que lhe devolverá o nome do programa a ser executado bem como seus parâmetros. A execução se processará através da chamada Runpgm, passando as informações recebidas. Quando se tratar de Procshell, o programa sequencial em questão será o S-Shell. Neste caso, antes de ser disparado o módulo Executor, se houverem parâmetros, os mesmos terão de ser substituidos.

Após o término de sua tarefa (Back, Prog ou ProcShell), o processo Envelope executará a instrução Start, possibilitando, dessa forma, a execução de novos programas seqüenciais (caso o último tenha sido descontinuado por uma instrução Stop). Chamará então o procedimento Estadoproc.Wakeup para acordar seu pai, que poderá estar aguardando pelo término de sua execução, e, finalmente, "morrerá", executando o procedimento Estadoproc.Exit.

3. CONCLUSÃO

O sistema SONIX deverá ser compatível com o UNIX, também, a nível do sistema de arquivos. Entretanto, como a implementação do sistema de arquivos projetada estava um tanto atrasada, decidiu-se incorporar ao SONIX o sistema de arquivos do sistema SOLO. Is-

to permitiu colocar em funcionamento em protótipo da SONIX no minicomputador LABO-8034. Este protótipo está operacional desde março do corrente ano.

O programa Pascal Concorrente que implementa o sistema SONIX possui cerca de 1500 linhas de código fonte. Foram efetuados testes metódicos e exaustivos, o que permite ter-se uma grande confiança na correção do seu funcionamento.

Paralelamente com os testes e com a implementação do sistema de arquivos estão sendo preparados os utilitários do sistema. Desta forma espera-se que em um pequeno espaço de tempo se possa colocar à disposição dos usuários em sistema completo, com grandes facilidades para o desenvolvimento de suas aplicações.

A experiência adquirida no desenvolvimento deste trabalho permite que se enfatize a conveniência de usar-se linguagens de alto nível no desenvolvimento de sistemas operacionais. A implementação destes sistemas complexos fica enormemente facilitada com a utilização de uma linguagem adequada.

BIBLIOGRAFIA

/AVR 83/ AVRITZER, Alberto. µzunix um sistema operacional com suporte para multiprocessamento e tempo compartilhado. Belo Horizonte, UFMG, 1983. (dissertação de mestrado).

/BOU 78/ BOURNE, S.R. The Unix Shell. The Bell System Technical Journal, 57(6): 1971-90, July-Aug. 1978.

/BRA 84/ BRANDÃO, Maria de Fátima Ramos. Especificação de um sistema operacional multi-usuário em Pascal Concorrente. Porto Alegre, CPGCC/UFRGS, 1984. (dissertação de mestrado).

/CAR 81/ CARVALHO, M. A. de. Sistema Operacional Duo. Porto Alegre, CPGCC/UFRGS, 1981. (não publicado)

/COS 83/ COSTA, C.M.; BARBOSA, L.F.; FRIEDRICH, L.F.; SAYÃO, M. & CORSO, T.B. Descrição da implementação de um sistema operacional multiprogramado, escrito em Pascal Concorrente. IN: CONFERÊNCIA LATINOAMERICANA de INFORMÁTICA, 10., Vina Del Mar, Abril 1984. Anales. Vina del Mar, CLEI, 1984.

/DEI 84/ DEITEL, Harvey M. An introduction to operating system. Reading, Addison-Wesley, 1984.

/GRA 79/ GRAEF, N.; KRETSCHMAR, H. LOER, K.P. & MORAWETZ, B. How design and implement small time-sharing systems using Concurrent Pascal. Software-Practice and Experience, 9(1):17-24, jan. 1979.

/HAN 77/ HANSEN, Per Brinch. The architecture of concurrent programs. Englewood Cliffs, Prentice-Hall, 1977.

/HOL 83/ HOLT, R.C. Concurrent Euclid, Unix System And Tunis. Reading, Addison-Wesley, 1984.

/ICH 83/ ICHIHARA, Ana Travassos. Projeto de um sistema de memória virtual para a máquina pascal concorrente emulada no computador ED-311. Porto Alegre, CPGCC/UFRGS, 1983. (Dissertação de mestrado)

/JOH 78/ JOHNSON, S.C. & RITCHIE, D.M. Portability of C programs and the Unix system. The Bell System Technical Journal, 57(6):2021-48, July-Aug. 1978.

/KRU 82/ KRUIJER, H.S.M. A Multi-user operating system for transaction processing, writen in Concurrent Pascal. Software-Practice and Experience, 12(5): 445-54, May 1982.

/MED 81/ MEDEIROS, Gil Carlos Rodrigues. Implementação do sistema Pascal Concorrente no computador Labo-8034. Porto Alegre, CPGCC/UFRGS, 1981. (dissertação de mestrado)

/RIT 74/ RITCHIE, D.M. & THOMPSON, K. The Unix time-sharing system. Communications of the ACM, 17(7):365-75, July 1974.

/RIT 78/ RITCHIE, D.M. & THOMPSON, K. The Unix Time-Sharing System. The Bell System Technical Journal, 57(6): 1905-29, July-Aug. 1978.

/RIT 78a/ RITCHIE, D.M. Unix Time-Sharing System: A Retrospective. The Bell System Technical Journal, 57(6):1947-69, July-Aug. 1978.

/THO 78/ THOMPSON, K. Unix Implementation. The Bell System Technical Journal, 57(6):1931-46, July-Aug.1978.

/WEB 82/ WEBER, T.S. & ROCHA, A.C. Descrição da arquitetura da máquina Pascal Concorrente. Porto Alegre, CPGCC/UFRGS, 1982. (relatório interno).

C.M. DA COSTA*, S.S. TOSCANI**, T.B. CORSO***

SUMÁRIO

O artigo descreve a implementação da linguagem de comandos S-Shell do sistema operacional SONIX, que é compatível com a linguagem Shell do sistema operacional UNIX. São apresentadas todas as características da linguagem, sua gramática, os analisadores léxico e sintático, o gerador de código e o interpretador desse código.

ABSTRACT

This work describes the implementation of the command language S-Shell of the SONIX operatings system that is a subset os the UNIX shell language. The characteristics of the language, its grammar, the lexical and syntathical analysers, the code generator and the interpreter for this code are presented.

*B.SC. Análise de Sistemas (PUC-RS, 1978), aluno de mestrado em Ciência da Computação (UFRGS), professor assistente do Departamento de Ciências Estatísticas e da Computação (UFSC); áreas de interesse: sistemas operacionais e linguagens de programação; Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99 - Porto Alegre - RS - CEP 90.000.

** Engenheiro Eletricista (UFRGS, 1967), Mestre em Informática (PUC/RJ, 1969); áreas de interesse: programação concorrente e sistemas operacionais; professor adjunto do Departamento de Informática/Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99 - Porto Alegre - RS - CEP 90.000.

*** Engenheiro Civil (UFRGS, 1973), Mestre em Ciência da Computação (UFRGS, 1978); áreas de interesse: programação concorrente e sistemas operacionais; Professor Adjunto do Departamento de Informática/Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99 - Porto Alegre - RS - CEP 90.000.

1. INTRODUÇÃO

S-Shell é a linguagem de comandos que serve de interface entre o usuário e o sistema Sonix. Esta linguagem é um subconjunto da linguagem Shell do Unix que mantém todas as principais características daquela.

Sua implementação foi feita por um programa escrito em Pascal Seqüencial. Este programa fica permanentemente sendo executado pelos processos Console do sistema operacional /COS 85a/.

A implementação da linguagem S-Shell foi realizada através da definição de uma máquina virtual (MV) que interpreta um conjunto de instruções apropriado para a implementação da linguagem S-Shell.

Os comandos dos usuários são transformados em instruções da máquina virtual, as quais são armazenadas em uma área de memória para posterior interpretação.

O interpretador de comandos foi estruturado em dois módulos, denominados Analisador e Executor.

No módulo Analisador são implementados os procedimentos necessários à análise léxica, a análise sintática e à geração de código para a máquina virtual. No módulo Executor foi implementado o procedimento que faz a interpretação do código gerado pelo módulo Analisador.

Os comandos executados pelo S-Shell podem ser oriundos em de terminal ou de um arquivo em disco. No segundo caso o usuário interativo, operando em um terminal, comanda a execução de uma outra ocorrência do programa S-Shell, redirecionando sua entrada para um arquivo construído previamente. Esta forma de uso do interpretador de comandos é denominada de "Procedimento S-Shell".

Quando o sistema Sonix é carregado ele cria um processo Console para cada terminal existente no sistema. Por sua vez, cada processo Console carrega para execução uma ocorrência do programa S-Shell, o qual passa a interagir com o usuário.

No momento em que o usuário digita uma linha de comando, esta linha é analisada e compilada para o código da máquina virtual. Se não é detectado nenhum erro, é disparado o processo de interpretação.

O S-Shell não é dotado de nenhum mecanismo de recuperação de erro. Quando for constatada a ocorrência de um, será enviada uma mensagem elucidativa para o usuário, todas as variáveis e estruturas de dados serão reinicializadas e o programa estará apto a aceitar um novo comando.

2. CONSTRUÇÕES DO S-Shell

O interpretador de comandos S-Shell possui as seguintes construções da linguagem Shell do Unix/RIT 78/:

Comandos Simples
Lista de comandos
Redirecionamento de entrada e saída
Linhas de montagem
Comando If
Comando While
Procedimentos Shell
Comandos em "background"

Algumas destas construções foram um pouco restringidas. Do comando If foi eliminada a cláusula Elseif; nos procedimentos Shell a passagem de parâmetros por assinalamento direto foi eliminada, sendo mantida a passagem de parâmetros posicionais. Com referência aos comandos em "background", somente é permitido que o caracter "&" que o indica apareça no final de um comando (simples ou lista de comandos). Assim, todos os comandos que precedem a tal caracter, no caso de existir mais de um, serão executados em background.

Em relação às variáveis Shell, permaneceram somente as que possuem significado implíci

to e que são mantidas e atualizadas pelo próprio Shell, e as que armazenam informações importantes para o usuário e que podem ser atualizadas pelo mesmo. As variáveis cujos conteúdos podem ser alterados pelo usuário são inicializadas pelo sistema no momento em que uma sessão é começada.

As variáveis anteriormente referidas são as seguintes:

Variáveis con Significado Implícito e Atualizáveis pelo Shell

$? - Contém o valor retornado pelo último comando executado. Este valor será zero se a execução tiver ocorrido normalmente e um no caso contrário.

$# - Armazena o número de parâmetros posicionais fornecidos quando a presente versão do Shell foi invocada.

$$ - Contém o número do processo associado a este Shell.

$! - Contém o número do último processo disparado no modo "background".

Variáveis Inicializadas no Momento em que Começa uma Sessão e Atualizáveis pelo Usuário

$Mail - Contém o nome do arquivo no qual o usuário deverá receber suas mensagens.

$Home - Contém o nome do diretório ao qual o usuário pertence.

$Path - Possui a lista de diretórios nos quais deverão ser procurados os arquivos contendo código executável. Se não forem especificados pelo usuário, "/Bin" e "/User/Bin" serão pesquisados. Na implementação do S-Shell é permitido ao usuário especificar até três diretórios.

As seguintes construções do Shell não foram implementadas:

- Comando Case
- Comando For
- Documentos locais
- Concatenação de saída
- Geração de nomes de arquivos
- Abolição do significado de metacaracteres

Alguns aspectos importantes, do ponto de vista de utilização, da linguagem S-Shell implementada são os seguintes:

a. Um comando simples pode ter um redirecionamento de entrada ("<"), um redirecionamento de saída (">"), ou ambos.

b. Nos comandos simples que fazem parte de uma linha de montagem ("|") não se pode ter no comando à esquerda do símbolo "|" uma alteração de saída e no comando à direita do "|" uma alteração de entrada.

c. Não pode haver redirecionamento múltiplo de entrada e saída.

d. O caracter "&", indicativo de "background", deve aparecer no final de um comando. Assim, por exemplo, em comandos if o caracter deve suceder ao Fi e em comandos While deve suceder ao Done.

3. GRAMÁTICA DA LINGUAGEM S-SHELL

A gramática será descrita com o uso da seguinte convenção:

- Os não terminais são escritos em letras maiúsculas.
- Os terminais são escritos em letras minúsculas.
- Uma produção será representada por um não terminal seguido de dois pontos e um conjunto de terminais e não terminais separados por brancos.
- Se várias produções começam pelo mesmo não terminal, serão postas em linhas separadas e o não terminal de origem da produção aparecerá somente uma vez.

```
ITEM : PALAVRA
       ENTRADA-SAÍDA

COMANDO-SIMPLES : ITEM
```

COMANDO-SIMPLES ITEM

```
COMANDO: COMANDO-SIMPLES
         (LISTA-DE-COMANDOS)
         While LISTA-DE-COMANDOS
         Do LISTA-DE-COMANDOS
         Done
         If LISTA-DE-COMANDOS
         Then LISTA-DE-COMANDOS
         Else LISTA-DE-COMANDOS
         Fi
         ATRIBUIÇÃO

LINHA-DE-MONTAGEM : COMANDO
                    LINHA-DE-MONTAGEM | COMANDO
EOU     : LINHA-DE-MONTAGEM
          EOU && LINHA-DE-MONTAGEM
          EOU || LINHA-DE-MONTAGEM

LISTA-DE-COMANDOS : EOU
                    LISTA-DE-COMANDOS;
                    LISTA-DE-COMANDOS&
                    LISTA-DE-COMANDOS; EOU

ENTRADA-SAÍDA : < PALAVRA
                >   PALAVRA

ATRIBUIÇÃO : NOME = PALAVRA

NOME : "seqüência de letras  começando por "$""

PALAVRA : "seqüência de letras e dígitos"

LETRAS : a, ..., z. $, /,-

DIGITOS : 0..9
```

As seguintes palavras são reservadas na linguagem:

If Then Else fi While Do Done

Os seguintes caracteres tem significado especial:

| Símbolo de linha de montagem
&& Símbolo de "E" lógico
|| símbolo de "OU" lógico
; Separador de comandos
& Comandos em "background"
() Agrupamento de comandos
< Redirecionamento de entrada
> Redirecionamento de saída

IMPLEMENTAÇÃO

A descrição da implementação do interpretador de comandos S-Shell será feita nas seções a seguir. Inicialmente será descrito o módulo Analisador e posteriormente o módulo Executor.

4.1 Módulo Analisador

O módulo Analisador é responsável pelo procedimentos de análise léxica, análise sintática e geração de código.

O analisador léxico opera sob demanda do analisador sintático, esse construído com o uso do método descendente recursivo.

Análise Léxica

A análise léxica é feita com o uso de seis procedimentos, os quais interagindo produ-

zem os elementos léxicos da linguagem, denominados "tokens". Os "tokens" são representados na seguinte estrutura de dados:

```
Type
    Classechar = (Letra, Dígito, Delimitador);
    Tipotoken  = Record
                      Classe  : Classechar;
                      Símbolo : Integer
                 End;
```

Os campos deste registro possuem o significado descrito a seguir.

Classe

Representa a classe a que o "token" pertence. Pode ser Letra, Dígito ou Delimitador.

Símbolo

Identifica o "token". Quando a classe a que o "token" pertence for letra, pode assumir os valores Palavra, Cif, Cthen, Celse, Cfi, Cwhile, Cdo ou Cdone. Quando a classe a que o "token" pertence for Dígito, assume o valor Número. Quando a classe for Delimitador pode assumir os valores Igual, E-comercial, Dois-ecom, Barra, Duas-bar, Ponto-virg, Abrepar, Fecha-par, Menor, Maior, Dois-pontos, Cr, Eof. Tratando-se de Palavra ou Número, o valor lido, que representa o nome de um comando ou o parâmetro para algum comando, encontra-se armazenado no vetor Nome.

A seguir serão apresentados os procedimento que implementam a análise léxica com a descrição da função que cada um desenvolve, individualmente.

Procedure Getchar;

Tem a função de ler um caracter de um dispositivo de entrada de dados e classificá-lo quanto a sua classe (Letra, Dígito ou Delimitador).

Procedure Classepal;

Verifica se a palavra contida no vetor Nome é uma palavra reservada da linguagem. Esta verificação é feita com o uso do comando Case.

Se a letra que ocupa a primeira posição no vetor Nome for inicial de alguma palavra reservada, então é verificado se o conteúdo do referido vetor é igual a esta específica palavra reservada. Se o resultado do teste for verdadeiro, significa que foi identificada uma palavra reservada. Neste caso a constante identificada por seu nome, acrescida do prefixo "C", será atribuído ao campo símbolo do registro que representa o "token".

Caso o resultado do teste seja negativo, o campo símbolo do referido registro receberá o valor Palavra, pois se está diante de um nome de comando ou nome de parâmetro.

Procedure Lepalavra;

Possui a função de ler uma seqüência de caracteres de um dispositivo de entrada de dados e armazená-los no vetor Nome.

Este procedimento é chamado sempre que, quando invocado pelo Léxico, o procedimento Getchar atribuir ao campo Classe do registro que representa o "token" o valor Letra, indicando com isto que uma letra pertencente ao alfabeto foi lida pelo mesmo.

Procedure Lenumero;

Sua função é idêntica à executada pelo procedimento Lepalavra, com a diferença de que é invocado para armazenar no vetor Nome uma seqüência de dígitos, os quais representam um número.

Procedure Saltabranco;

Este procedimento tem a função de saltar caracteres branco através de sucessivas chamadas ao procedimento Getchar.

Procedure Léxico;

É o procedimento básico de todo o processo de análise léxica. É chamado pelo analisador sintático sempre que esse necessita de um novo "token".

Através de sua ação, são disparados, apropriadamente, os demais procedimentos necessários para obter e tornar disponível um novo elemento léxico.

Análise Sintática

O analisador sintático é o gerenciador da interpretação das requisições de serviço feitas pelos usuários. Comanda a ação do analisador léxico, invocando-o sempre que necessita de um "token", e executa o atendimento da solicitação, através da chamada do Interpretador ao término da análise de um comando.

Na sua construção foi empregado o método descendente recursivo, o que permitiu explorar uma característica importante da linguagem Pascal, qual seja, a recursividade.

O código virtual gerado pelo analisador sintático é colocado em uma área de memória acessível ao interpretador (área de comunicação entre o analisador sintático e o interpretador). Todas as informações necessárias ao interpretador, geradas pelo analisador sintático, estão contidas no código, eliminando, desta forma, a necessidade de acessos a outras estruturas de dados.

Os procedimentos que implementam à análise sintática podem ser classificados em "Procedimentos de Análise e Geração de Código" e "Procedimentos Auxiliares".

Procedimentos de Análise e Geração de Código

Os procedimentos que se enquadram nesta categoria atuam de forma semelhante. Quando o usuário digita um comando ou quando o interpretador de comandos é acionado para executar comandos contidos em um arquivo, o programa S-Shell invoca o analisador léxico, o qual devolve um "token" da linguagem.

A partir deste "token" e, com o auxílio de um comando Case, a ação seguinte no processo de análise e geração de código é definida com a obtenção de um novo "token".

A geração de código é feita apropriadamente por alguns destes procedimentos através do procedimento Gera, o qual é invocado da seguinte maneira:

Gera (Mnem, Oper1 : Integer; Oper2: Identificador);

Os procedimentos anteriormente referidos são os seguintes:

Procedure Comando;
Procedure Listadecomandos;
Procedure Comandosimples;
Procedure Continuaçãodocomando;
Procedure Fimdocomando;
Procedure Instalaparametro;
Procedure Alteraentrada;
Procedure Alterasaída;
Procedure Atribuição;
Procedure Linhademontagem;
Procedure Andf;
Procedure Orf;

Procedimentos Auxiliares

Os procedimentos ditos Auxiliares são os que desenvolvem algum outro tipo de função necessária ao processo de análise e geração de código. Tais procedimentos são os seguintes:

Procedure Initvar;

Possui a função de inicializar todas as variáveis e tabelas, sempre que inicia o processo de interpretação de um novo comando.

Procedure Execcomandos;

É o procedimento que dispara o processo de interpretação. Fica permanentemente executando, somente deixando de fazê-lo quando o usuário encerra o S-Shell, através do comando Exit.

Procedure Atribui-par (Var a : Univ Identificador; i : Integer);

Atribui um parâmetro ao campo operando de uma instrução da MV. Sua existência é necessárias pois as variáveis Param a Instrução são de tipos diferentes.

Procedure Change-par;

A função desenvolvida por este procedimento é a de efetuar a substituição dos parâmetros posicionais pelos respectivos argumentos de chamada, fornecidos pelo usuário ao invocar a execução de um Procedimento S-Shell.

O usuário, quando está utilizando o sistema, o está fazendo através de uma ocorrência do programa S-Shell.

O S-Shell é um programa seqüencial como qualquer outro utilitário. Isto possibilita que o usuário comande a execução de uma nova ocorrência do mesmo, redirecionando sua entrada de dados para um arquivo criado anteriormente que contenha comandos do S-Shell. A isto denomina-se de Procedimento S-Shell.

Esta nova ocorrência do programa interpretador de comandos será executada por um novo processo.

Este processo, por sua vez, receberá do processo pai os parâmetros, se existirem, e os tornarão disponível e alteráveis ao programa seqüencial S-Shell através da declaração

Program P (var Param : Regparam)

suprida no prefixo do sistema Pascal.

A definição do tipo Regparam é a seguinte:

```
Type
    Identificador = Array (.1..20.) of Char;
    Regparam = Array (.0..9.) of Identificador;
```

Quando, durante a fase da análise e geração de código, o S-Shell identifica um "token" Eof, isto significa que o mesmo está atuando como um Procedimento S-Shell. Neste caso, antes de disparar a execução do módulo Executor, providência a substituição dos parâmetros posicionais por seus respectivos argumentos, invocando o procedimento Change-par, em questão.

Procedure Background;

Sua invocação se dá no momento em que é identificado um "token" Ecomercial.

A função desenvolvida por este procedimento é a de transferir o código virtual gerado até aquele momento para o monitor Controlproc, do sistema operacional Sonix, de onde o código será retirado para ser interpretado por um processo especialmente criado para este fim.

Com isto, o controle retorna imediatamente para o usuário que não precisará aguardar pelo término da execução das solicitações de serviço feitas anteriormente, podendo passar de imediato a enviar novos comandos.

Procedure Gera (Mnem, Operi : Integer; Oper2 : Identificador);

Sua função é gerar código para a MV. Recebe como parâmetros o mnemônico e o operando e produz uma nova instrução no vetor código.

4.2 Módulo Executor

As instruções geradas pelo módulo Analisador são executadas pelo módulo Executor.

A MV concebida para a implementação da linguagem de comandos, é constituída por uma memória que contém o código virtual gerado e por um Interpretador, responsável pela exe-

cução das instruções.

O Interpretador é invocado pelo procedimento "Comandos", pertencente ao analisador Sintático, sempre que um "token" Cr ou Eof, indicador de fim de comando do usuário, for encontrado.

Conjunto de Instruções

As instruções da MV são descritas a seguir:

JMP

O apontador de instruções virtuais (Pc) recebe o conteúdo do campo operando da instrução.

FALSEJMP

Pc recebe o conteúdo do campo operando da instrução se o último comando disparado não executou corretamente. Esta informação é obtida através da chamada do procedimento Resultexecp (I), pertencente ao sistema operacional.

COMMAND

Indica que um comando, cujo identificador está armazenado no campo operando da instrução, deve ser executado.

CPARAM

Contém, em seu campo Operando, o parâmetro do comando especificado em uma instrução Command. São geradas tantas destas instruções quantos forem os parâmetros fornecidos para o comando.

ALTERIN

A entrada de dados do comando a ser executado deve ser redirecionada para o arquivo especificado no campo operando desta instrução.

ALTEROUT

A saída de dados do comando deve ser redirecionada para o arquivo especificado no campo operando desta instrução.

CPIPE

Indica que deve ser estabelecido um canal de intercomunicação entre dois processos.

ASSIGN

Indica que deverá ser feita uma atribuição à uma variável S-Shell, do tipo atualizável pelo usuário, especificada no campo operando desta instrução.

VALUE

Deve suceder a uma instrução Assign. O conteúdo de seu campo operando é o valor que deve ser atribuido à variável especificada na última instrução Assign.

CWAIT

O processo que a executa fica bloqueado a espera que um processo filho termine. Isto é conseguido através da chamada do procedimento Waitp, pertencente ao sistema operacional.

No momento em que o processo filho termina sua execução, o processo que executou a presente instrução é sinalizado, o que ocasiona o seu prosseguimento. Esta instrução não possui operando.

CSTOP

Esta instrução ao ser executada faz com que o interpretador termine e o controle retorne ao módulo Analisador. A mesma não possui operando.

Exemplos de Geração de Código

```
Comando : Ls
Código Virtual Gerado : Command Ls
                        Cwait
```

```
                        Cstop
```

```
Comando : Cdr; Ls
```

```
Código Virtual Gerado : Command Cdr
                        Cwait
                        Command Ls
                        Cwait
                        Cstop
```

```
Comando : Ls > Meuarq
```

```
Código Virtual : Command Ls
                 Alterout Meuarq
                 Cwait
                 Cstop
```

```
Comando : Ed fonte < Arq
```

```
Código Virtual : Command Ed
                 Cparam Fonte
                 Alterin Arq
                 Cwait
                 Cstop
```

```
Comando : If Spascal Arqfonte Arqobjeto > Printer Then Arqobjeto
          Else
              Editor Arqfonte
          Fi
```

```
Código Virtual : 1 Command Spascal
                 2 Cparam Arqfonte
                 3 Cparam Arqobjeto
                 4 Alterout Printer
                 5 Cwait
                 6 Falsejmp 10
                 7 Command Arqobjeto
                 8 Cwait
                 9 Jmp 13
                10 Command editor
                11 Cparam Arqfonte
                12 Cwait
                13 Cstop
```

```
Comando : If Cdr Meudir
          Then Ls
          Fi
```

```
Código Virtual : 1 Command Cdr
                 2 Cparam Meudir
                 3 Cwait
                 4 Falsejmp 7
                 5 Command Ls
                 6 Cwait
                 7 Cstop
```

```
Comando : Ls | Who
```

```
Código Virtual : Command Ls
                 Cpipe
                 Command Who
                 Cwait
                 Cstop
```

```
Comando : Spascal Fonte Objeto && Objeto
```

```
Código virtual : 1 Command Spascal
                 2 Cparam Fonte
```

```
                3 Cparam Objeto
                4 Cwait
                5 Falsejmp 8
                6 Command Objeto
                7 Cwait
                8 Cstop
```

Comando : Spascal Fonte Objeto > Printer || Editor Fonte

```
Código Virtual : 1 Command Spascal
                 2 Cparam Fonte
                 3 Cparam Objeto
                 4 Alterout Printer
                 5 Cwait
                 6 Falsejmp 8
                 7 Jmp     11
                 8 Command Editor
                 9 Cparam Fonte
                10 Cwait
                11 Cstop
```

```
Comando : While Produztexto < Ent > Texto Do
          I     Imprimetexto < Texto > Printer
                Done
```

```
Código Virtual : 1 Command Produztexto
                 2 Alterin Ent
                 3 Alterout Texto
                 4 Cwait
                 5 Falsejmp 11
                 6 Command Imprimetexto
                 7 Alterin Texto
                 8 Alterout Printer
                 9 Cwait
                10 Jmp 1
                11 Cstop
```

Interpretador

A interpretação do código virtual gerado pelo módulo Analisador é feito por um procedimento denominado Interpretador.

Tal procedimento é formado por um comando de repetição (While) e por um comando de seleção (Case).

O comando While faz com que o procedimento permaneça em excução até que seja encontrada uma instrução Stop, que o encerra. No comando Case, cada rótulo identifica uma instrução e possui o procedimento necessário à sua execução.

Existem alguns comandos que, pela freqüência de uso, devem residir permanentemente na memória principal. Para tal, estes comandos foram codificados em um procedimento denominado Execbuiltin. Assim, no momento em que o Interpretador identifica que tem um comando para executar, ele deve primeiramente verificar se não se trata de um comando codidicado internamente. Esta verificação é feita através da invocação do procedimento Builtin. A técnica de pesquisa empregada é a mesma referida na descrição da análise léxica, para a identificação de palavras reservadas (utiliza-se o comando Case para a pesquisa em questão).

Acrescentar ou retirar comandos codificados internamente é algo trivial, pois significa incluir ou excluir entradas no comando Case e seus respectivos procedimentos.

Os comandos que foram codificados internamente são os seguintes:

Exit - Termina a execução do programa S-Shell.

Ps - Lista os processos existentes no sistema e os nomes dos programas que cada um está

tá presentemente executando.

Kill - Matá o processo especificado por seu número de criação (o qual é fornecido como argumento) e todos os seus descendentes.

5. CONCLUSÃO

O programa que implementa a linguagem S-Shell possui cerca de 1500 linhas de código fonte.

Sua depuração ainda não foi totalmente concluída porque os procedimentos que implementam redirecionamento de entrada, redirecionamento saida e "pipe" atuam diretamente nas tabelas que compõe o sistema de arquivos, e o mesmo encontra-se ainda em fase de desenvolvimento.

Do ponto de vista de utilização, a linguagem mostra-se fácil de ser usada e fácil de ser aprendida. Como a linguagem oferece ótimos recursos aos usuários, espera-se que a mesma venha a se constituir em uma ferramenta importante, no auxílio ao desenvolvimento de programas.

BIBLIOGRAFIA

/AVR 83/ AVRITZER, Alberto. µzunix um sistema operacional com suporte para multi-Processamento e tempo compartilhado. Belo Horizonte, UFMG, 1983. (dissertaçao de mestrado).

/BOU 78/ BOURNE, S.R. The Unix Shell. The Bell System Technical Journal, 57(6): 1971-90, July-Aug. 1978.

/BRA 84/ BRANDÃO, Maria de Fátima Ramos. Especificação de um sistema operacional multi-usuário em Pascal Concorrente. Porto Alegre, CPGCC/UFRGS, 1984. (dissertaçao de mestrado).

/CAR 81/ CARVALHO, M.A. de. Sistema Operacional Duo. Porto Alegre, CPGCC/UFRGS, 1981. (não publicado)

/COS 83/ COSTA, C.M.; BARBOSA, L.F.; FRIEDRICH, L.F.; SAYÃO, M. & CORSO, T.B. Descrição da implementação de um sistema operacional multiprogramado, escrito em Pascal Concorrente. IN: CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA, 10., Viná Del Mar, Abril 1984. Anales. Viná del Mar, CLEI, 1984.

/COS 85/ COSTA, C.M. SONIX - Um sistema operacional multi-usuário escrito em Pascal Concorrente. Porto Alegre, CPGCC-UFRGS, 1985. (dissertaçao de mestrado).

/COS 85a/ COSTA, C.M.; TOSCANI, S.S. & CORSO, T.B. SONIX - Uma implementação do Unix em Pascal Concorrente. Porto Alegre, CPGCC/UFRGS, 1985 (Não publicado).

/DEI 84/ DEITEL, Harvey M. An introduction to operating system. Reading, Addison-Wesley, 1984.

/GRA 79/ GRAEF, N.; KRETSCHMAR, H.; LOER, K.P. & MORAWETZ, B. How design and implement small time-sharing systems using Concurrente Pascal. Software-Practice and Experience, 9(1): 17-24, Jan. 1979.

/HAN 77/ HANSEN, Per Brinch. The architecture of concurrent programs. Englewood Cliffs, Prentice-Hall, 1977.

/HOL 83/ HOLT, R.C. Concurrent Euclid, Unix System And Tunis. Reading, Addison-Wesley, 1984.

/ICH 83/ ICHIHARA, Ana Travassos. Projeto de um sistema de memória virtual para a máquina pascal concorrente emulada no computador ED-311. Porto Alegre, CPGCC/UFRGS, 1983. (Dissertaçao de mestrado).

/JOH 78/ JOHNSON, S.C. & RITCHIE, D.M. Portability of C programs and the Unix system. The Bell System Technical Journal, 57(6):2021-48, July-Aug. 1978.

/KRU 82/ KRUIJER, H.S.M. A Multi-user operating system for transaction processing, writen in Concurrent Pascal. Software-Practice and Experience, 12(5): 445-54, May 1982.

/MED 81/ MEDEIROS, Gil Carlos Rodrigues. Implementação do Sistema pascal Concorrente no computador Labo-8034. Porto Alegre, CPGCC/UFRGS, 1981. (dissertação de mestrado).

/RIT 74/ RITCHIE, D.M. & THOMPSON, K. The Unix time-sharing system. Communications of the ACM, 17/7):365-75, July 1974.

/RIT 78/ RITCHIE, D.M. & THOMPSON, K. The Unix-Sharing System. The Bell System Technical Journal, 57(6):1905-29, July-Aug. 1978.

/RIT 78a/ RITCHIE, D.M. Unix Time-Sharing System: A Retrospective. The Bell System Technical Journal, 57(6):1947-69, July-Aug. 1978.

/THO 78/ THOMPSON, K. Unix Implementation. The Bell System Technical Journal, 57(6):1931-46, July-Aug. 1978.

/WEB 82/ WEBER, T.S. & ROCHA, A.C. Descrição da arquitetura da máquina Pascal Concorrente. Porto Alegre, CPGCC/UFRGS, 1982. (relatorio interno).

SISTEMA DE AQUISIÇÃO E EXIBIÇÃO DE SINAIS ANALÓGICOS PARA MICROCOMPUTADORES (1)

G. M. STEPHAN* e W. LOBIANCO JUNIOR**

SUMÁRIO

Foi desenvolvido um sistema que permite a um microcomputador nacional adquirir e exibir sinais analógicos simultaneamente ao seu processamento. A aquisição dos sinais, com frequência de amostragem programável, é feita por interrupção, permitindo processamento em tempo-real. Para a exibição de sinais existem controles para um traçador XY, um registrador x-t e um monitor de raios catódicos o qual permite a visualização instantânea dos sinais amostrados e do processamento em tempo-real.

O Sistema pode ser conectado ao barramento de qualquer microcomputador com CPU-Z8Ø. Foram desenvolvidas rotinas básicas de controle e Programas aplicativos sob o sistema operacional CP/M.

As aplicações do sistema se estendem desde o processamento de sinais biológicos até o controle de processos.

ABSTRACT

An acquisition and exhibition system of analogue signals for microcomputers has been developed, assembled and tested. The acquisition interface has a programable timer which interrupts the CPU allowing real-time processing with programable sampling rate. The interface provides control for a XY plotter, a strip chart recorder and a CRT monitor, which allows an immediate visualization of the sampled signals and results of real-time processing.

The interface can be plugged directly to the bus of any Z8Ø based microcomputer. A set of control routines and utilitaries has been developed under CP/M for the system.

The applications are wide including biologic signal processing and process control.

* Técnico em eletrônica (CEFET-RJ, 1981), estudante do 4º ano de engenharia eletrônica na UFRJ, manutenção, projeto, construção de circuitos ligados a microprocessadores, COPPE/UFRJ, PEB, CAIXA POSTAL 68510 CIDADE UNIVERSITÁRIA, CT - RIO DE JANEIRO, RJ.

** Estudante do 5º ano de engenharia eletrônica na UFRJ, Programador ASSEMBLY, FORTRAN, BASIC E PASCAL, Estagiário da SHELL BRASIL S/A (PETRÓLEO).

Trabalho orientado por Fernando Soares Schlindwein; Professor Assistente do Programa de Engenharia Biomédica da COPPE/UFRJ, Tese de Mestrado em 1982.

(1) - 1º PRÊMIO IV CONCURSO DE TRABALHOS DE INICIAÇÃO CIENTÍFICA

I. INTRODUÇÃO

Entendemos por Interface Analógica o conjunto de circuitos que permitem a transformação de sinais analógicos em informações compreensíveis por um computador digital e vice-versa.

Tal interface, quando controlada por um pacote de "software", forma o que chamamos de Sistema de Aquisição e Exibição de Sinais Analógicos.

I.1 - Importância de um Sistema de Aquisição de Sinais

Computadores digitais não têm a capacidade de trabalhar em domínios contínuos, pois a maneira destes representarem números é discreta. A maioria dos sistemas físicos são observáveis através de funções contínuas no tempo (as quais passaremos a chamar de sinais). A fim de processarmos tais sinais (entende-se por processar um sinal como o ato de fazer qualquer cálculo que envolva a descrição matemática do sinal, tal como calcular suas derivadas, integrais, compará-lo com padrões, calcular sua transformada de Fourier ou filtrá-lo) em computadores digitais necessitamos discretizar o sinal tanto no domínio (tempo) como no contradomínio (tensão) através de um método de amostragem.

A amostragem é realizada medindo-se valores da função em determinados instantes de tempo (5).

Isto pode ser feito graficamente, com o auxílio de um registrador x-t ou de um osciloscópio de memória. A partir do sinal desenhado são feitas medidas e estas são passadas ao computador, via teclado.

O procedimento descrito acima é muito trabalhoso, pouco preciso e não permite um processamento em tempo-real.

Quando desejamos que o processamento seja em tempo-real ou com maior precisão devemos usar um sistema de aquisição de sinais. Com tal sistema conseguimos uma precisão nas medidas proporcional ao número de bits do conversor A/D e o processamento pode ser feito em tempo-real pois as medidas são colocadas imediatamente a disposição do computador.

I.2 - Importância de um Sistema de Exibição de Sinais

Durante a amostragem, é interessante termos alguma informação, em tempo-real, sobre como o sinal está sendo amostrado e processado. Para que isto seja possível torna-se necessário um sistema de exibição de sinais rápido, que tome pouco tempo do computador para não atrasar o processamento em tempo-real.

É interessante também, obtermos gráficos em papel, resultantes da aquisição e do processamento dos sinais.

II. DESCRIÇÃO DO "HARDWARE"

O "hardware" do sistema é apresentado em diagrama de blocos na Figura 1.

Os blocos correspondentes a amplificadores, filtros e ajuste de tensão de nível médio representam os circuitos contidos na caixa cinza vista em cima do osciloscópio nas figuras 3 e 5. Tais circuitos bem como os transdutores dependem do sistema a ser observado e não serão descritos.

Os blocos envoltos pela linha tracejada representam os circuitos montados em uma placa de "wire-wrap" alojada dentro de um microcomputador SDE-42 fabricado pela EMBRACOMP (figuras 3 e 4). Tais blocos são descritos a seguir e todos os comentários entre parênteses devem ser acompanhados na Figura 2.

II.1 - Aquisição

Foram utilizados oito circuitos de "sample and hold", permitindo assim a amostragem simultânea dos oito canais analógicos. O acionamento destes circuitos é feito no mesmo instante em que ocorre um pedido de interrupção para a CPU. Quem determina este instante é um temporizador programável (8253) que gera a frequência de amostragem [1, 4, 6].

Uma vez interrompida, a CPU pode comandar o início da conversão analógica-digital de qualquer um dos oito canais através de uma instrução de "OUT". O fim da conversão é detectado por consulta a um "bit" de uma porta através de uma instrução de "IN". Este processo leva cerca de 150 micro segundos, o que limita a frequência de amostragem máxima, para um só canal, em 6 MHz.

II.2 - Exibição

Como formas de exibição de sinais temos a capacidade de:

- Desenhar 4 gráficos de 256 por 256 pontos, com alta velocidade, em tubo de raios catódicos (CRT) com entradas XY.
- Registrar figuras de 256X256 pontos em traçador XY com controle da pena.
- Registrar dois gráficos com 256 níveis de ordenada em registrador x-t.

II.2.1 - Saída para CRT-XY

O circuito de exibição rápida em CRT-XY é composto pelo contador (4040), multiplexador (3X74LS157), "buffer 3-state" bidirecional (74LS245), memória de exibição (2X2114), conversor D/A de 8 "bits" (DAC 08), multiplexador analógico (4051, aqui funcionando como conversor D/A de 2 "bits") e gerador de rampa sincronizado [1, 2, 3, 4, 9].

Desde que o microprocessador não esteja acessando a memória de exibição, é o contador que gera seu endereçamento. O oitavo "bit" desse contador sincroniza o gerador de varredura que fornece o eixo X do CRT. O eixo Y é uma tensão proporcional ao conteúdo da posição de memória endereçada e ao bloco de 256 "bytes" acessado.

Os 1024 pontos são exibidos no monitor, organizados em 4 linhas de 256 pontos cada. Para essa organização os oitavo e nono bits do contador são usados para endereçar um multiplexador analógico (4051) com quatro tensões DC à entrada, que são somandas analogicamente ao conteúdo da memória de exibição [7].

Para monitorar um sinal que está sendo amostrado, basta que a rotina de processamento em tempo-real inclua uma subrotina que coloque a última amostra na memória de exibição, preenchendo-a como um "buffer" circular. Com essa técnica podem-se monitorar até 4 sinais analógicos amostrados simultaneamente colocando um em cada linha. Os próprios resultados do processamento (derivada, detecção de eventos, ...) podem ser exibidos no monitor em tempo-real, usando-se uma linha para cada informação.

II.2.2 - Saída para Traçador XY e Registrador x-t

O circuito de controle do Traçador é o mesmo do registrador x-t. Sendo assim, só podemos usar uma dessas formas de exibição de cada vez, conforme indicado na figura 1.

Esse circuito é composto pelas portas A e B da 8255, por dois conversores D/A (DAC 08) e um bit da porta C da 8255 (PC2) [1, 4].

Quando utilizamos o traçador, alocamos o conversor da porta A para o eixo-X e o da porta B para o eixo-Y. Desta forma, com instruções de OUT para essas portas, posiciona-se a pena nas coordenadas desejadas. O controle de elevamento e abaixamento da pena é feito por PC2.

Quando utilizamos o registrador x-t, alocamos um conversor D/A para cada canal e PC2 para controlar o acionamento do papel, determinando assim quando registrar os sinais.

Observações:

i) Para microcomputadores com interface paralela tipo "Centronics" foi desenvolvido um "software" capaz de copiar os 4 gráficos exibidos no CRT-XY em impressora com capacidade gráfica no padrão "EPSON". O resultado da cópia pode ser observado nas Figuras 5 e 6.

ii) Podemos, com o circuito de controle do plotter e um sensor óptico acoplado à PC7, digitalizar, em alto contraste, figuras em preto e branco, com uma resolução de 256X X256 pontos [8].

iii) Apesar de a interface ter sido projetada para funcionar conectada ao microcomputador SDE-42, ela pode ser adaptada a qualquer microcomputador que tenha como CPU o microprocessador Z80. As modificações são mínimas e, em uma versão em circuito impresso, podem ser feitas através de "jumpers".

Esta facilidade de adaptação é consequência de 3 fatos:

1) Termos todos os dispositivos da interface mapeados em entrada/saída permitindo, assim, o seu uso em microcomputadores que possuem toda a sua capacidade de endereçamento de memória ocupada (64k).

2) A decodificação de endereços ser completa, feita na própria placa da interface e reconfigurável.

3) Existirem "drivers" para todas as linhas de entrada na placa, podendo ser inversores ou não-inversores.

III. DESCRIÇÃO DO SOFTWARE

Foram desenvolvidas rotinas básicas para uso de plotter XY, monitor de raios catódicos (ou osciloscópio XY), manipulação de arquivos em disquete e programas que lidam com amostragem.

Todo programa que utilizar a interface deve chamar a rotina de inicialização, que tem como função preparar as portas do C.I. 8255 adequadamente.

Para uso do traçador, foram desenvolvidas rotinas que permitem: levantar, abaixar, posicionar e obter a localização da pena; fixar as coordenadas (0,0) e (255, 255), que caracterizam o campo no qual a figura vai ser desenhada; traçar e graduar eixos ortogonais

em escala linear ou logarítmica; ligar dois pontos por interpolação linear, e copiar um canal do monitor em papel.

No que concerne ao monitor e manipulação de arquivos existem rotinas capazes de mover pontos de uma região da tela para outra, ler um arquivo de dados gravado em disquete e exibí-lo na tela do monitor, bem como gravar alguns ou todos os pontos visualizados na tela em um arquivo em disquete.

Para realizar amostragens, o sistema possui rotinas que permitem escolher a frequência de amostragem via teclado, inicializar o contador programável (8253) e liberar ou inibir os pulsos de interrupção para o microprocessador.

Além dessas rotinas básicas, foi desenvolvido um programa aplicativo utilizando todas as rotinas anteriormente descritas, permitindo 9 funções distintas, as quais são mostradas nas Figuras 5 e 7.

As rotinas básicas formam uma biblioteca. Tal biblioteca tem sido usada por alunos de pós-graduação para elaboração de programas aplicativos.

IV. RESULTADOS

Foi desenvolvido um sistema capaz de: 1) Amostrar até oito sinais analógicos com frequência de amostragem programável até 750Hz quando amostrando 8 canais e 6KHz quando amostrando apenas um canal; 2) Exibir em tubo de raios catódicos com entradas XY esses sinais ou resultados de processamento em tempo-real, por acesso direto a memória, economizando, assim, tempo de CPU; 3) Registrar os sinais amostrados, resultados de processamento ou qualquer figura de 256 X 256 pontos em traçador, incluindo controle de elevamento e abaixamento da pena; 4) Registrar os sinais amostrados, resultado de processamento ou qualquer gráfico com 256 níveis de ordenada em registrador x-t, incluindo o controle de acionamento do papel; 5) Digitalizar uma figura em alto contraste com uma resolução de 256 X 256 com auxílio de um plotter e um sensor óptico; 6) Trabalhar conectada ao barramento de qualquer microcomputador com CPU-Z80 e sistema operacional CP/M.

V. DISCUSSÃO

O sistema descrito apresenta como vantagens o baixo custo e grande versatilidade para análise de sinais em tempo-real tais como: eletrocardiograma, eletroencefalograma, velocidade de fluxo sanguíneo ou respiratório, temperaturas e pressões em caldeiras, etc..

Tendo em vista a falta de padronização dos conectores de barramentos dos diversos microcomputadores, a instalação da interface analógica requer um trabalho de fiação particular para cada microcomputador. Para maior facilidade e versatilidade de instalação, é possível implementar-se um controlador baseado em microprocessador que comande a interface, transformando-a em um periférico inteligente capaz de ser conectado a qualquer computador através de uma interface padrão (RS 232 C, por exemplo).

O sistema tem tido boa aceitação de seus usuários no Programa de Engenharia Biomédica da COPPE/UFRJ e está em funcionamento sem falhas desde sua conclusão em dezembro de 1984.

BIBLIOGRAFIA

1. Component Data Catalog, Intel Corporation, January, 1981.

2. JOSEPH J. CARR (1980), Z80 Users Manual.

3. GANDRA, S.A.T., (1982), "Monitor de Arritmias Cardíacas em tempo-real utilizando o intervalo R-R e a largura do QRS", Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro, RJ.

4. Linear Databook, National Semiconductor Corporation, 1982.

5. OPPENHEIM, A.V.; SHATER, R.W., (1975) "Digital Signal Processing". Prentice-Hall Inc.

6. SCHLINDWEIN, F.S., (1982), "Microcomputador para Análise de Sinais de Fluxo Sanguíneo Arterial Captados por Ultra-Som Doppler", Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro, RJ.

7. SCHLINDWEIN, F.S.; CAPRIHAN, A.; GANDRA, S.A.T., (1983), "Sistema de Aquisição e Exibição de Sinais Biológicos para Microcomputador", Trabalho publicado na Revista Brasileira de Engenharia - Caderno de Engenharia Biomédica, Vol. 1, Nº 2.

8. SCHLINDWEIN, F.S.; PIRES, C.A.C.; RAMOS, M.P., (1983), "Digitalizador bidimensional utilizando registrador X-Y", Comunicação Científica publicada no Catálogo de temas curtos do VIII Congresso Brasileiro de Engenharia Biomédica.

9. Semiconductor Data Library CMOS, vol. 5, séries B, Motorola Semiconductor Product Inc., 1976.

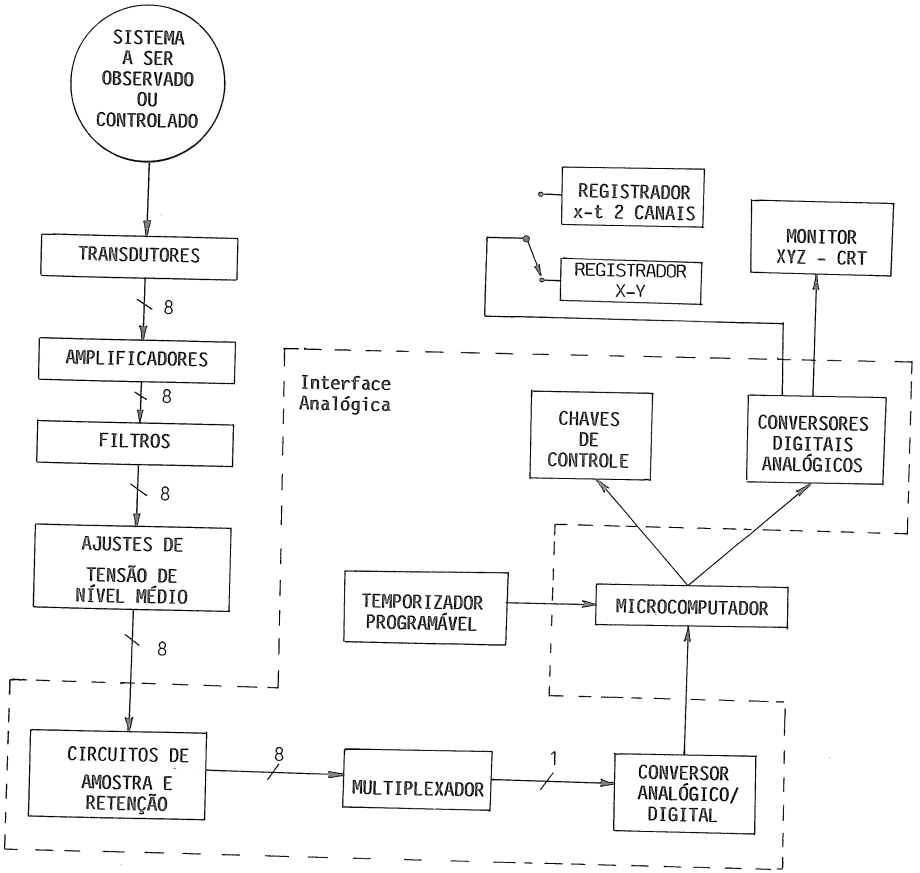
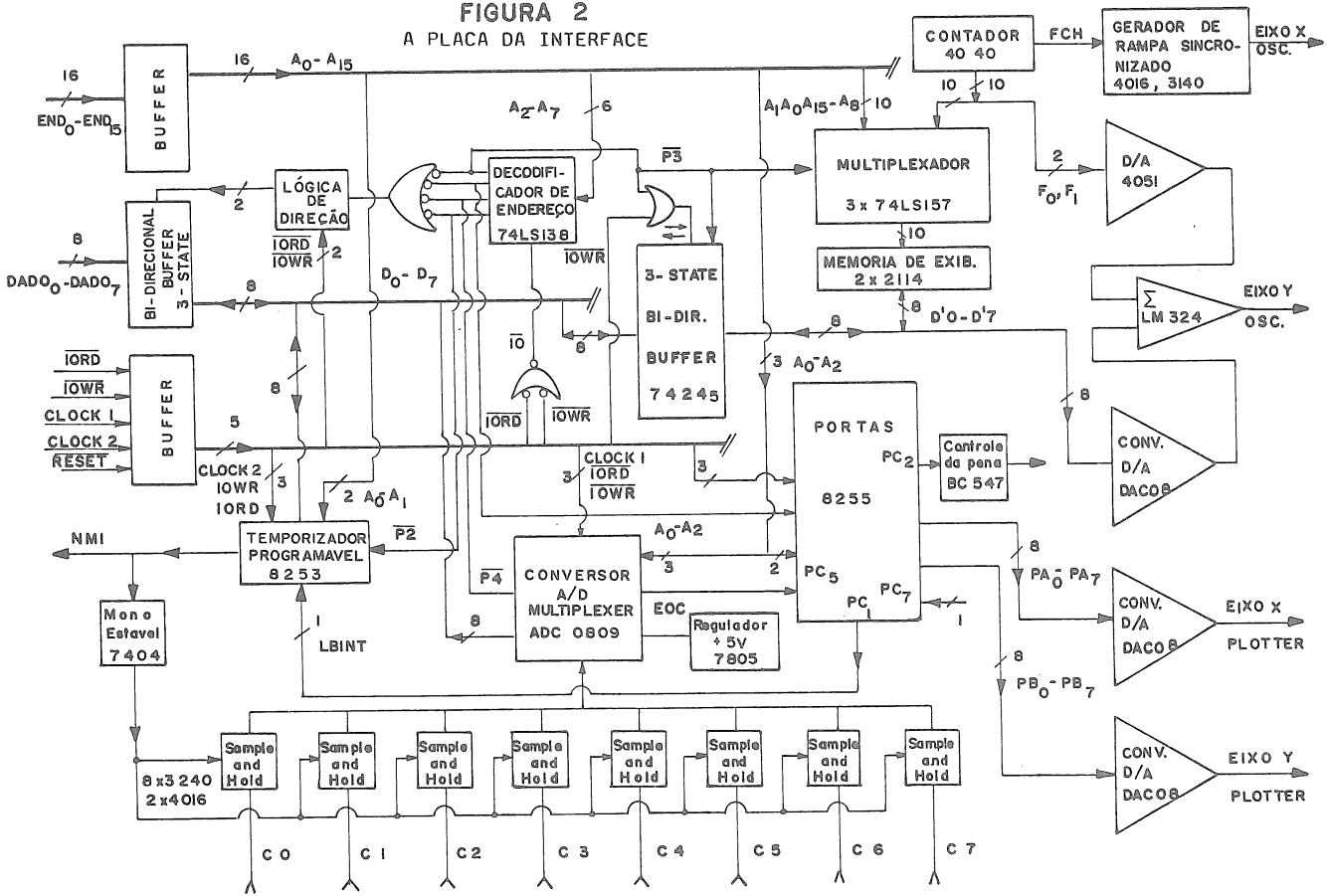


Figura 1 - Diagrama de blocos do "hardware" do sistema de aquisição e exibição de sinais analógicos.

FIGURA 2

A PLACA DA INTERFACE



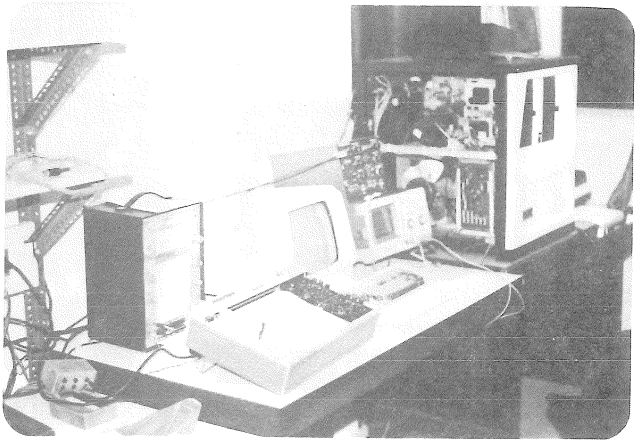


FIG.3 - Sistema completo

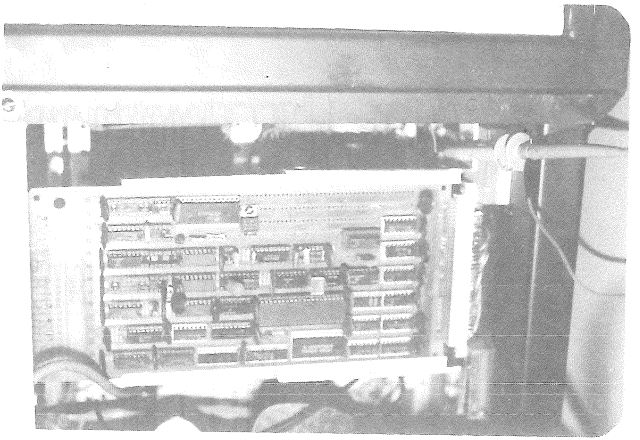


FIG. 4 - Detalhe da placa da interface analógica.

FIG. 5 - Osciloscópio mostrando conteúdo de um arquivo

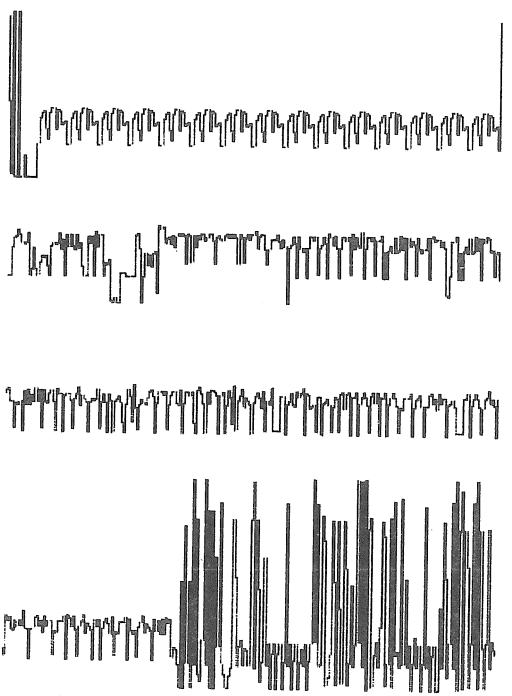


FIG. 6 - Cópia do sinal exibido no osciloscópio

```
A>MENU

SISTEMA DE AQUISICAO E EXIBICAO DE SINAIS ANALOGICOS
======= == ========= = ======== == ====== ==========
P R O G R A M A S   D I S P O N I V E I S :
 1. AMOSTRA SINAL E EXIBE NA TELA
 2. LE ARQUIVO DO DISQUETE E EXIBE NA TELA
 3. MOVIMENTA TRECHOS NA TELA
 4. CRIA ARQUIVO EM DISQUETE CONTENDO PONTOS DA TELA
 5. PERMITE FIXAR O ZERO E O FUNDO DE ESCALA DO PLOTTER
 6. DESENHA EIXOS ORTOGONAIS
 7. GRADUA OS EIXOS DESENHADOS NO PLOTTER
 8. TRACA SEGMENTOS DE RETA NO PLOTTER
 9. COPIA UM CANAL DA TELA NO PLOTTER

ENTRE COM O NUMERO DO PROGRAMA ESCOLHIDO: 1

QUAL A FREQUENCIA DE AMOSTRAGEM EM HERTZ? (1HZ A 5KHZ) ^C
A>
A>
```

FIG. 7 - MENU

SISTEMA DE PULVERIZAÇÃO CATÓDICA PARA DEPOSIÇÃO DE FILMES FINOS DE DIÓXIDO DE SILÍCIO (1)

C. R. Peter*

SUMÁRIO

Foi realizado o projeto e a construção de um sistema para deposição de filmes finos de dióxido de silício, a partir da pulverização catódica de silício em descarga elétrica contínua em ambiente com pressão reduzida (~ 100 m Torr) contendo argônio e oxigênio. A análise dos filmes obtidos, por retroespalhamento elástico de Rutherford com partículas alfa evidencia que estes têm a estequiometria do dióxido de silício, sendo esta homogênea em todo seu volume. Este sistema de deposição está em operação corrente com desempenho bastante satisfatório, tanto pela simplicidade e estabilidade de operação como pela repetibilidade dos resultados. Um segundo protótipo mais versátil, é descrito nesta publicação.

ABSTRACT

We performed the design and building of a deposition system for preparation of silicon dioxide thin films from DC sputtering of silicon in atmosphere with reduced pressure (~ 100 m Torr), containing argon and oxigen. By Rutherford Backscattering analysis with alpha particles of the obtained films it was shown that the estequiometry was the one of the

(1) 2º PRÊMIO IV CONCURSO DE TRABALHOS DE INICIAÇÃO CIENTÍFICA

silicon dioxide and homogeneous in the all volume. 693

The deposition system, now in normal operation presents characteristics like simplicity and stability of operation and repeatibility of the results. A second more versatile prototype is described in this publication.

*Estudante de graduação em Física, UFRGS; bolsista de iniciação científica pela PROPESP para o grupo de Microeletrônica; Instituto de Física, Av. Luiz Englert, s/nº, sala 215-B, Porto Alegre, RS.

INTRODUÇÃO

Os filmes finos de dióxido de silício (SiO_2), pelas suas propriedades elétricas e químicas são amplamente empregados nos processos de construção de microcircuitos. As características destes filmes permitem sua utilização principalmente como barreira contra penetração de impurezas nos processos de difusão e implantação iônica e como dielétrico. Nos estudos de implantação iônica, após a implantação de determinado elemento, é necessário que se faça um recozimento do material implantado para reorganização da rede cristalina. O recozimento consiste num processo térmico que ocorre a temperaturas elevadas (500 a 1200 ^{o}C para o silício). Para evitar que durante o recozimento ocorra difusão para fora das impurezas implantadas, utiliza-se como barreira um filme de SiO_2 convenientemente depositado. [1]

Existem vários processos para deposição de filmes de SiO_2 sobre silício. Os processos mais comumente utilizados são: deposição química a partir de fase vapor (CVD) [2] e a pulverização catódica em radiofrequência (RF Sputtering). [3,4]

O processo de deposição química (CVD) propicia a formação de SiO_2 a partir da decomposição térmica da silana (SiH_4) e a oxidação do silício resultante. Obtém-se altas taxas de deposição e excelentes características elétricas. Porém o reagente principal (SiH_4), além do custo muito elevado, é de difícil obtenção no mercado.

A técnica de preparação de filmes de SiO_2 por pulverização catódica em descarga RF é bastante difundida, porém requer equipamento relativamente complexo e catodo de SiO_2.

Um processo menos frequentemente reportado para formação de SiO_2 é o que emprega pulverização catódica reativa do si

lício em descarga DC em atmosfera de pressão reduzida (~ 100 m Torr) contendo argônio e oxigênio. Este processo requer equipamentos mais simples e o material do cátodo pode ser facilmente obtido em alta pureza com lâminas usuais para microeletrônica . Neste processo de preparação do SiO_2, os átomos de silício arrancados do cátodo pelo bombardeamento iônico reagem com o oxigênio presente na descarga formando o SiO_2 que se deposita sobre o ânodo.

O controle da espessura do filme depositado pode ser feito através da corrente e tempo de duração do processo para uma determinada tensão de descarga.

DESENVOLVIMENTO DO PROTÓTIPO

O protótipo inicial mostrado esquematicamente na figura 1, consiste de uma câmara cilíndrica de pirex de 150 mm de diâmetro, com tampa de alumínio, montado sobre um sistema de alvo vácuo que permite bombeamento de seu interior até pressões de 10^{-7} Torr. No interior dessa câmara foram dispostos dois eletrodos, respectivamente cátodo e ânodo. O cátodo é uma lâmina de silício de 57 mm de diâmetro e 400 μm de espessura em contato mecânico e elétrico com um disco de alumínio de 63 mm de diâmetro e 6 mm de espessura. Um anel de alumínio de diâmetro externo de 63 mm e interno de 54 mm com 2 mm de espessura, aparafusado contra o disco de alumínio, mantém o cátodo de silício preso ao disco.

Este conjunto tem contato elétrico com o exterior por meio de um passador de cerâmica (feed-through) o qual está afixado na tampa da câmara de pirex. Envolvendo o conjunto e dis

tante dele de 2 mm temos uma blindagem eletrostática aterrada , construída em aço inox de 2 mm de espessura. Um orifício de 54 mm de diâmetro permite que o cátodo seja exposto à descarga elétrica. Entre o cátodo e a blindagem não se forma descarga pois sendo a largura da região escura maior que o distanciamento entre estes dois elementos, não existe possibilidade de formação de plasma. Assim sendo a blindagem previne que regiões do cátodo não cobertas por silício sejam pulverizadas o que implicaria em contaminação do filme depositado. Dessa forma permite-me que o bombardeio ocorra somente sobre a região do catôdo constituída pelo silício, evitando contaminações do filme por materiais indesejáveis.

O ânodo é constituído por um suporte de amostras em alumínio, conectado ao terra e que pode ter sua distância em relação ao cátodo ajustada de maneira conveniente.

Os gases argônio e oxigênio são admitidos na câmara através de válvulas do tipo agulha, para permitir um controle fino da vazão. Primeiramente a câmara é evacuada até pressões de 10^{-7} Torr, quando então a válvula gaveta que separa a câmara do sistema de bombeamento de alto vácuo é parcialmente fechada. Introduz-se então oxigênio e argônio na câmara; no nosso estudo foram empregadas pressões parciais de 30 m Torr de oxigênio e 100 m Torr de argônio. Ao se aplicar um potencial negativo ao cátodo que nosso caso é de 1 kV observa-se a formação de uma descarga luminosa entre o cátodo e ânodo com a formação de uma região escura de cerca de 5 mm de espessura junto ao cátodo. Com uma descarga de 1 kV e 10 mA (0,5 $mA.cm^{-2}$)obtém-se uma velocidade de deposição de SiO_2 de 15 Å por minuto, sobre um substrato de silício de 38 mm de diâ

metro, distante 2,0 cm do cátodo. Observa-se boa uniformidade de espessura numa área central com cerca de 25 mm de diâmetro (com decréscimo de 10% da espessura do filme nas bordas), verificada pela coloração do filme sob luz branca.

Estamos limitados a taxa máxima de deposição de 15 Å/min pela fonte de alta tensão disponível que fornece no máximo 10 mA e além disso o cátodo não é refrigerado para poder suportar maior potência na descarga.

A distância de 2,0 cm entre cátodo e ânodo é a que conferiu uma melhor uniformidade de deposição nas condições de funcionamento acima mencionadas. Convém ainda ressaltar que a taxa de deposição é repetitiva (dentro de ± 5%) e o funcionamento da descarga muito estável.

Análises por espalhamento Rutherford de partículas alfa [6] de 760 keV realizadas com o implantador iônico do Instituto de Física da UFRGS, mostram que se tem 2 átomos de oxigênio para cada átomo de silício em toda extensão do filme, sendo portanto SiO_2 estequiométrico e homogêneo. Essas análises também foram usadas para medida da espessura dos filmes e determinação da taxa de deposição.

A influência do oxigênio na taxa de crescimento dos filmes não pode ser precisamente determinada, pelo fato do argônio que utilizamos conter elevada taxa de umidade e pelo fato do sistema de vácuo não ser conveniente para uma degaseificação por aquecimento para diminuir a umidade. Verificamos que descargas somente com argônio produziam filmes de SiO_2 porém com a espessura abaixo da esperada quando se adiciona oxigênio.

Este sistema tem sido utilizado para deposição de filmes de 1000 Å de espessura sobre amostras previamente implantadas,

com o objetivo de evitar difusão para fora de dopantes durante os processos de recozimento térmico. [7]

PROJETO DE UM SISTEMA MAIS VERSÁTIL

Com base nas experiências que obtivemos com este sistema de pulverização catódica, elaboramos o projeto de um sistema mais versátil o qual está atualmente em fase final de construção. Esse novo sistema possui dois cátodos de 100 mm de diâmetro, refrigerados por circulação de água. O ânodo (suporte de amostras) consiste de um disco de alumínio que pode ser rodado por acionamento externo de forma que se possa trocar de amostras sem novo ciclo de bombeamento. A câmara de vácuo é feita em aço inox e as vedações com anel de viton, de forma que se possa realizar a degaseificação do sistema por aquecimento até 150 ^{o}C. A figura 2 apresenta um corte central desse conjunto, onde temos as seguintes especificações: 1 - cátodo refrigerado; 2 - blindagem eletrostática; 3 - ânodo, suporte dos substratos; 4 - anteparo; 5 - visor; 6 - câmara de vácuo em inox 304; 7 - isolador em teflon.

Com a refrigeração do cátodo é possível operar com potências maiores, o que permite a obtenção de taxas de deposição mais elevadas. Esperamos com esse novo sistema obter espessura uniforme sobre área com cerca de 70 mm de diâmetro.

O argônio será admitido na câmara após um processo de purificação utilizando limalha de cobre aquecida a 300 ^{o}C, sílica gel e peneira molecular de 4 Å. O oxigênio será purificado pela passagem através da sílica gel e peneira molecular de 4 Å. [8]

No início do processo a descarga será realizada entre o anteparo e o cátodo, possibilitando que a superfície do cátodo seja limpa pela erosão por alguns minutos. Após isto o anteparo será girado por comando externo para que ocorra deposição sobre os substratos.

Esse novo sistema, devido tanto à purificação dos gases quanto à possibilidade de degaseificação da câmara poderá ser utilizado também para deposição de filmes metálicos ou de seus compostos (óxidos, nitretos, etc...). Estudos posteriores empregando novo protótipo, vizarão estudar a influência de parâmetros como pressão parcial do oxigênio sobre a estequiometria do óxido formado e taxa de deposição, assim como influência de parâmetros da descarga como tensão, densidade da corrente, pressão do argônio sobre uniformidade e taxa de deposição.

Além disso também serão realizados estudos para caracterizar eletricamente o filme produzido, quanto a rigidez dielétrica, contaminação iônica e densidade com objetivo de se determinar sua viabilidade no emprego como camada passivadora sobre dispositivos eletrônicos integrados.

O equipamento construído mostrou-se adequado à preparação de filmes de SiO_2. Trata-se de equipamento bastante simples e de baixo custo, porém de alta repetibilidade e estabilidade de funcionamento. Utilizando-se gases comerciais como argônio e oxigênio e lâmina de silício para material do cátodo, realizamos processo de pulverização catódica reativa com consequente preparação de filmes de SiO_2 que tem-se prestado como barreira contra difusão de dopantes para fora de amostras em processos de recozimento de danos de implantação iônica. Ressaltamos no sistema construído a importância da blindagem eletrostática envolvendo o cátodo para evitar contaminação do filme por consequência da pulverização de materiais que não sejam silício. O equipamento descrito representa uma contribuição simples e acessível para laboratórios que necessitem preparar filmes de SiO_2 que são úteis também em áreas diversas da microeletrônica, como em ótica, proteção de superfícies, etc.

[1] S.R. Wilson et allii, "Rapid Isothermal Anneal of ^{75}As Implanted Silicon", Appl. Phys. Lett, 41 (10) pg. 978-980, 1982.

[2] J. Holleman, J. Middelhoek, "Report on the Silox Process", Technische Hogeschool Twente, Holand, 1970.

[3] L. Holland, Vacuum Deposition of Thin Films, Chapman & Hall, London, 1958, 541 p.

[4] V. Bruk, V. Garschermin e A. Kursonov. "Produccion de Dispositivos Semicondutores", Editora MIR, Moscow, 1971.

[5] B. Chapman, "Glow Discharge Processes, Sputtering and Plasma Etching", John Wiley & Sons, New York, 1980.

[6] W-K. Chu et allii, "Backscattering Spectrometry", Academic Press, New York, 1978, 384 p.

[7] C.M. Hasenack, J.P. de Souza e I.J.R. Baumvol, "The Suppression of Residual Defects in Silicon Implanted with Arsenic by Rapid Isothermal Annealing", a ser publicado em Nucl. Instr. and Meth., seção B.

[8] A.L. Kohl and F.C. Riesendfeld, "Gas Purification", 3 ed., Gulf Publishing, Houston, 1979, 825 p.

FIGURA 1

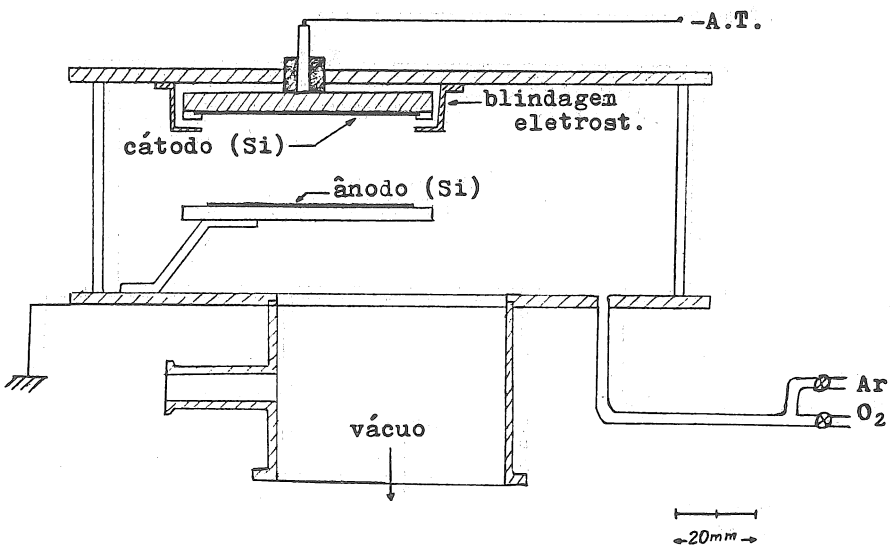
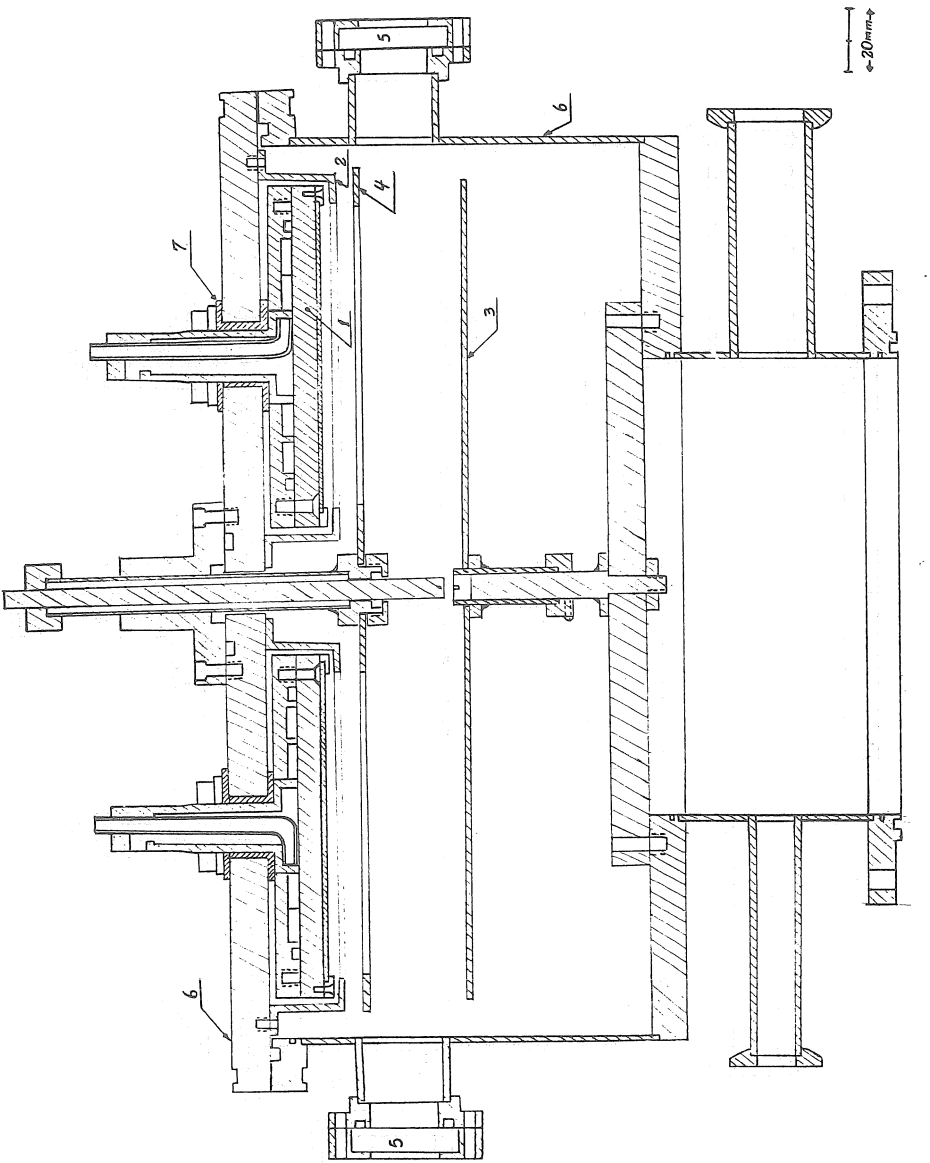


FIGURA 2



INTERFACE PARA LIGAÇÃO DE TRAÇADOR GRÁFICO A SISTEMA DE COMPUTAÇÃO [1]

T.I.NUNES*, C.T.SILVEIRA*, C.A.PROLO* e A. GUIGNATTI*
R.G.TELICHEVESKY**

SUMÁRIO

Este trabalho trata do projeto e implementação de uma interface, utilizando microprocessador Z80, para ligação de um traçador gráfico (plotter 1627 - Sistema IBM 1130) a um sistema de computação (Burroughs B6700) via teleprocessamento através do padrão RS-232. O objetivo foi tornar o traçador compatível com qualquer sistema de computação, tanto microcomputador como sistema de grande porte, viabilizando a sua utilização com equipamentos que possuam interface padrão V24.

ABSTRACT

This paper is concerned with the design and implementation of an interface, based on the Zilog Z80 microprocessor, for the conection of a plotter device (presently being used in an IBM 1130 System) to a Burroughs B6700 mainframe. Using the RS232 standard, this work will permit the future use of this kind of plotter with any other computer system, from microcomputers to large systems.

* Estudante de Engenharia Elétrica; UFRGS - CPD - Núcleo de Desenvolvimento Tecnológico. Rua Ramiro Barcelos, 2574 - POA - RS - Fone: 31.2355 Ramal 53.

** Engenheiro Eletrônico (UFRGS, 1980), mestre em Ciência da Computação (UFRGS, 1984); UFRGS - CPD - Núcleo de Desenvolvimento Tecnológico. Rua Ramiro Barcelos, 2574 - POA - RS - Fone: 31.2355 - Ramal 53.

(1) 3º PRÊMIO IV CONCURSO DE TRABALHOS DE INICIAÇÃO CIENTÍFICA

INTRODUÇÃO

O projeto surgiu da necessidade de uma melhor utilização de um equipamento de entrada/saída, no caso o traçador gráfico, pertencente a um sistema que tende a se tornar absoleto.

Atualmente o sistema IBM 1130 se interliga com o sistema B 6700 através de RJE (REMOTE JOB ENTRY). O objetivo principal foi possibilitar a ligação do traçador gráfico diretamente ao B 6700 através de uma linha de comunicação deste, permitindo a sua ligação "on-line", ou seja, ligada diretamente ao processador de comunicação (DCP), em "batch", etc... [7]. O projeto foi orientado de modo a torná-lo o mais versátil possível.

Para a elaboração do projeto realizaram-se estudos do sistema 1130 e seu traçador [1]; do sistema B 6700, seu processador de comunicação e protocolo POLL-SELECT [7]; e dos microprocessadores disponíveis no mercado [2, 4, 5].
Com os conhecimentos oriundos destes estudos foi elaborado o projeto do interface baseado no microprocessador Z80.
O sistema constitui-se basicamente do traçador, da interface e sua console.
Após a elaboração do projeto foi construído e testado um protótipo da interface e sua console.
A interface propriamente dita pode ficar simultaneamente conectada ao sistema IBM 1130 e a outro sistema de computação que quiser utilizar o traçador. Determina-se o sistema que comandará o traçador através da "chave de seleção" instalada na interface.
Desta maneira continua existindo a possibilidade anterior de se utilizar o traçador como periférico do sistema 1130.
Pensando na versatilidade do projeto adotou-se interface padrão RS 232C (norma V24) [2] e loop de corrente, tornando-o assim compatível com qualquer sistema que possua estes padrões. Assim com uma mudança no programa de controle podemos ligá-lo a qualquer microcomputador, a um nodo de rede local, etc...

CARACTERÍSTICAS DO TRAÇADOR

O traçador é o Plotter 1627 da IBM [1] e possui 3 elementos móveis: Pena (P), Rolo (R) e Carro (C).
Os desenhos são produzidos por movimentos incrementais independentes ou combinados da pena, rolo ou carro.
A pena é montada sobre o carro dotado de movimento horizontal. A pena pode subir ou descer. O movimento vertical é obtido através da rotação do rolo onde está fixado o papel.

Estas funções são executadas com as seguintes características:

a) Passo dos eixos X e Y : 1/100"
b) Velocidade dos eixos X e Y : 300 passos/s (máx)
c) Velocidade do eixo Z (Pena) : 10 operações/s
d) Papel : Largura : 10"
Comprimento: 4m
e) Tempo de execução: Carro e Rolo - 3,4ms
Pena - 110ms

Para comandar estes 3 elementos o traçador tem 6 sinais de entrada:

- Pena para cima (PC)
- Pena para baixo (PB)
- Rolo para baixo (RB)
- Rolo para cima (RC)
- Carro para Direita (DC)
- Carro para Esquerda (CE)

Não possui capacidade de armazenamento de seqüência de informações. Para executar um passo de qualquer um dos 3 elementos (P, R, C) é necessário enviar ao traçador um pulso de $\pm$ 12V com duração de 50µs. Anteriormente, para a utilização do traçador, fazia-se um programa em Fortran, com subrotinas existentes, rodava-se no IBM 1130 e ele se encarregava de transmitir os comandos de movimentos incrementais para o traçador. Com a interface o programa é rodado em qualquer linguagem, no B 6700 ou qualquer outro processador (mini, micro, etc...), e enviado à interface em pacotes que serão decodificados e comandarão o traçador. Com isso a interface não necessita de dedicação exclusiva do processador, liberando-o para executar outras tarefas enquanto o pacote de informações, depois de decodificado, é enviado ao traçador.

INTERFACE

A interface propriamente dita pode ser dividida em 4 blocos:

- Interface Serial
- UCP e Memórias
- Console para sinalização e controle
- Acionamento dos elementos móveis (P, R, C)

conforme figura 1.

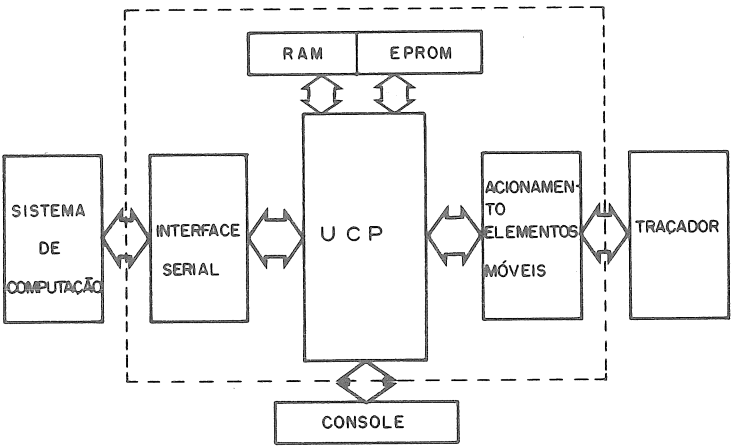


FIG. 1 - INTERFACE

UCP E MEMÓRIAS

O gerenciamento da interface é realizado por um microprocessador Z80 [5] operando na freqüência de 2,456MHz gerada por um oscilador à cristal.
A UCP é responsável pelo recebimento, transmissão e decodificação dos dados, e a partir destes, pelo controle do bloco de acionamento e sinalização da console.
O projeto foi feito possibilitando ao microprocessador acessar até 6 Kbytes de Eprom onde está o programa de controle do Z80, e 1 Kbyte de RAM estática, onde estará o Stack do Z80, bem como o buffer dos dados recebidos de sistema externo.

INTERFACE SERIAL

O elemento principal deste bloco, responsável pela transformação das mensagens recebidas ou transmitidas, de serial para paralelo e vice-versa, é uma USART (8251 da Intel) [4].
As mensagens poderão ser transmitidas ou recebidas nas velocidades 9600, 4800, 2400, 1200, 600 ou 110 Bps.
O protocolo já implementado, a ser usado na comunicação da interface com o B 6700 é o Poll-Select [7].

CONSOLE

Para as funções de controle e sinalização da interface foi necessária a implementação de uma console (ver figura 2).
A console visa facilitar a ação do operador e indicar, com sinais luminosos (LEDs) e sonoros (Bip), as ocorrências.
A console possui 4 botões conforme descrito abaixo:

- Inicia/continua: quando acionado indica que o traçador está preparado para iniciar ou continuar a operação (ex: papel e pena instalados ou recolocados, etc...)
- Reinicializa: quando acionado indica que o programa deve ser retransmitido, isto é, houve algum problema e o desenho deve ser feito de novo.
- Pára: quando acionado permite a intervenção do operador durante a execução de um programa.
- Reset do Bip: quando acionado desativa momentaneamente o Bip.

A console possui também uma sinalização luminosa, com Led's, indicando o seguinte:

- Parado
- Falta papel
- Final de sessão
- Recepção
- Cor da pena.

O traçador original não possui sensor de falta de papel, então foi instalado um dispositivo para esse fim, que sinaliza à interface quando isto ocorre. Visando a possibilidade de fazer um desenho colorido, foi implementado um indicador de cor da pena; quando aceso, um dos leds indica ao operador que deve trocar a pena.
O sinal sonoro indica alguma ocorrência fora do normal.

INTERFACE DE ACIONAMENTO DOS ELEMENTOS MÓVEIS

O papel principal desta interface é receber a informação de quais elementos devem ser acionados (P, R, C) e transformá-la nos sinais necessários para acionamento do traçador (pulsos de -12V com duração de 50μs).

SOFTWARE

Para controle do fluxo de informações, utilizou-se uma estrutura bastante simples. Existem 2 buffers de 256 caracteres na RAM da interface que conterão os comandos de plotagens.
O gerenciamento de fluxo é feito por mensagens de um byte transmitidas ao sistema externo. No início é enviado uma mensagem de "START" e o sistema externo manda 2 buffers de comandos. A partir daí, cada vez que um

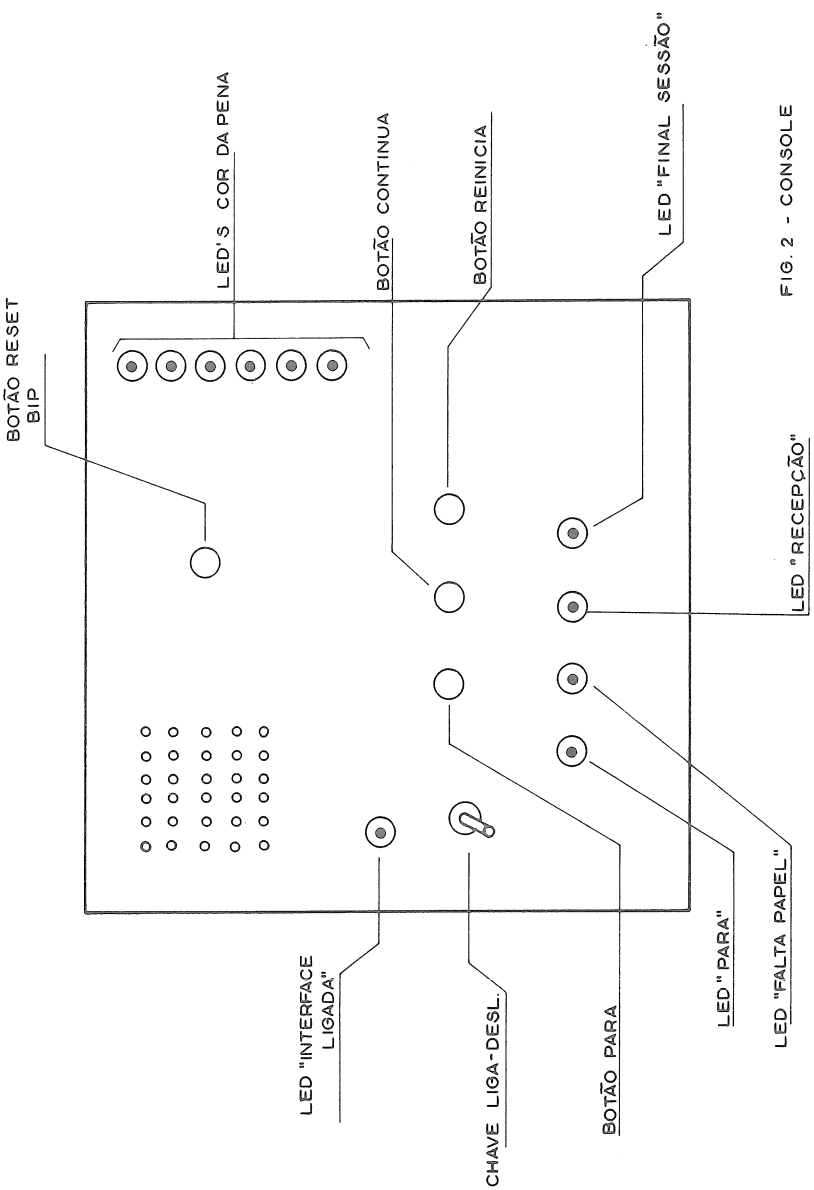


FIG. 2 - CONSOLE

dos buffers foi esgotado a interface passa a plotar o outro buffer e envia ao sistema uma mensagem de buffer vazio e assim por diante.
Isso permite que o traçador trabalhe sempre com carga máxima pois é ele o ponto de estrangulamento em termos de velocidade.
O software é então constituído basicamente de três blocos:

1. Atendimento
2. Manipulação dos dados recebidos e plotagem
3. Tratamento de exceções

1. Este bloco é basicamente uma máquina de estados, chamada por interrupção não mascarável pela USART (8251) [4] a cada caracter recebido.
Além de atender as requisições do protocolo Poll-Select, ela espera que a mensagem em si se inicie por dois caracteres especiais: Esc e Bel. Caso contrário a mensagem é descartada.
Isto permite que a interface funcione como um terminal comum de uma rede de teleprocessamento, sujeito a receber mensagens do sistema que de outro modo poderiam ser consideradas comandos de plotagem.

2. Este é o bloco encarregado de enviar os comandos ao traçador.
Cada byte vindo do equipamento externo conterá um comando para o traçador (pena + rolo + carro) ou um fator de repetição. Neste caso considera-se que o comando seguinte deverá ser repetido tantas vezes quantas determinar o fator de repetição.
Isto diminui sobremodo a quantidade de informação que transita pela linha de comunicação, pois os desenhos em geral contém grandes segmentos em reta.

3. O tratamento de exceções inclui inicialmente três componentes:

3.1 - Parada do desenho pelo operador. O operador tem a opção de interromper o desenho através de uma tecla da console. O desenho só será reativado após pressionar a tecla "continua" ou "reinicializa". No primeiro caso o desenho prossegue como se nada tivesse ocorrido. No segundo caso, o sistema externo é informado para retransmitir o desenho desde o início.

3.2 - Parada do desenho por falta papel - Há um sensor no traçador que ao detectar a falta interrompe o processador da interface. O procedimento é o mesmo que no caso anterior.

3.3 - Parada do desenho para troca de cor da pena - O byte de comando do traçador poderá solicitar troca de cor de pena. Neste caso, além do indicador sonoro e led de "parado", o led correspondente à cor ficará piscando até que o operador, depois de trocar a pena, pressione a tecla "continua".

CONCLUSÃO

Com a implementação do sistema interface-traçador obteve-se uma versatilidade antes inexistente, possibilitando sua ligação com uma gama enorme de sistemas e em várias condições. O projeto facilitou a operação do traçador, tanto na parte de controle como na sinalização.

A implementação dos dispositivos citados na descrição da console (sensor de falta de papel, indicador da cor da pena e sinalizador sonoro de falha) e a própria console melhoraram as características do traçador.

A console permite uma comunicação, antes inexistente, entre o operador e o sistema de computação. Desta maneira é mais fácil evitar possíveis problemas ou falhas na operação do traçador e também corrigí-las. Na realidade criou-se uma realimentação no sistema "sistema de computação-traçador" informando ao primeiro a situação do traçador.

O sensor de falta de papel impede que se continue um desenho quanto faltar ou rasgar o papel, evitando a perda do desenho e o dano à pena e ao rolo (pela inexistência de papel entre os dois). Estes problemas anteriormente eram de difícil solução.

O indicador de cor da pena permite ao usuário realizar um desenho com até 6 cores, possibilidade que sem esse dispositivo não existia.

Conseguiu-se assim uma melhor utilização do equipamento, tornando-o também acessível a um maior número de usuários sem esquecer o aumento na capacidade do traçador a um baixo custo.

BIBLIOGRAFIA

[1] IBM CORPORATION. 1627 Plotter Field Engineering Theory - Maintenance. San Jose, Cal, c 1963.

[2] GARLAN, H. Introduction to Microprocessor System Design. New York, McGraw Hill.

[3] TEXAS INSTRUMENTS INC. The TTL Data Book form Design Enginneers. Dallas, c 1984.

[4] INTEL CORPORATION. Intel Component Data Catalog. California, c 1979.

[5] ZILOG INC. Data Book 1982/83. California, c 1981.

[6] LEVENTHAL, L.A. Z80 Assembly Language Programming. New York, McGraw Hill, c 1981.

[7] BURROUGHS CORPORATION. B6700 Information Processing System. Detroit, c 1977.

COMPUTERS and PROGRAMMING LANGUAGES for AI

Dr Philip C. Treleaven

University of Reading
Reading RG6 2AX
England

Dr Isabel Gouveia Lima

Peat Marwick Mitchell Ltd.
Management Consultants
London

ABSTRACT

Since Japan launched its 10-year national Fifth Generation Computer project, the United States, the European Community, France, Germany and Britain have started similar national research programmes. Their aim is to develop a new generation of "intelligent" computers for use in the 1990s. Fifth Generation computers will be knowledge information processing systems designed to support the symbolic processing of artificial intelligence (AI). These national research programmes are leading to a rapid expansion in artificial intelligence, together with machine architectures and programming languages for AI. Machines for AI currently range from highly microprogrammed workstations specifically built for LISP or PROLOG, to novel parallel machines based on alternatives to control flow execution. Programming languages for AI also cover a spectrum from conventional procedural languages such as PASCAL, through symbolic languages such as LISP and PROLOG, to new "rule-based" expert system languages like OPS5. This paper presents an overview of machine architectures and programming languages specifically designed for artificial intelligence.

FIFTH GENERATION COMPUTERS

In October 1981 Japan launched its 10 year national Fifth Generation project to develop knowledge information processing systems and processors [17,26]. Since then other major industrial countries have started comparable national research programmes each with a significant proportion of funding devoted to artificial intelligence [7]. In the United States the Strategic Computing Initiative, a $600 million programme funded by the Defence Department, is investigating "machine intelligence technology that will greatly increase national security and economic power". In the European Community the ESPRIT programme has a significant part of its $1.3 billion funding devoted to AI technologies. The individual European countries are also funding major fifth generation programmes, for example Britain with $300 million funding and the Federal Republic of Germany with $1 billion funding. Lastly, in France there are a number of distinct research initiatives [7], some of which predate Japan's Fifth Generation project.

Each of the above national fifth generation programmes is broadly aimed at investigating AI technologies and developing symbolic processing computers for use in the 1990s. Many factors support the adoption of such a radically new generation of AI computers. Firstly, the handling of non-numerical data such as sentences, symbols, speech, graphics and images is becoming increasingly important. Secondly, the processing tasks performed by computers are becoming more "intelligent", moving from scientific calculations and data processing, to artificial intelligence applications. Thirdly, computing is moving from a sequential, centralised world to a parallel, decentralised world in which large numbers of computers are to be programmed to work together in computing systems. Lastly, today's computers are still based on the thirty-year-old von Neumann architecture; essentially all that has happened is that the software systems have been repeatedly extended to cope with the increasingly sophisticated applications.

The competition between the national fifth generation research programmes has been a catalyst for artificial intelligence research, and for the development of machine architecture and programming languages. This is illustrated by Figure 1.

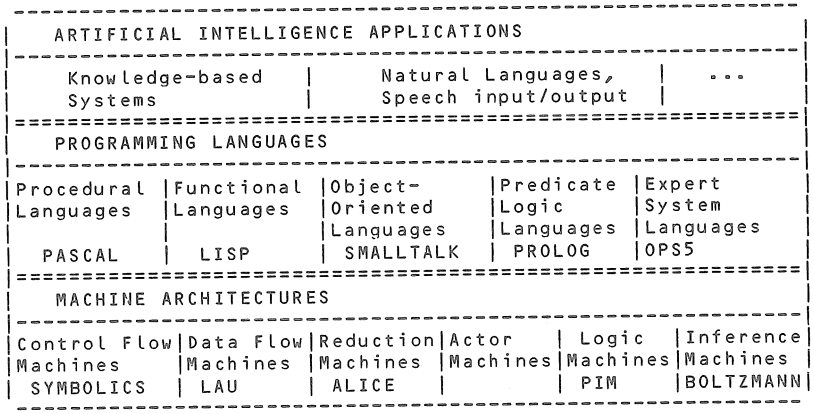


Figure 1: Machine Architectures and Programming Languages for Artificial Intelligence

The two major AI application areas are knowledge-based expert systems [11] and human oriented input-output, such as natural language and speech. For programming these applications, the major languages are LISP [31] and PROLOG [4,22], although specialised expert system languages such as OPS5 [8] are finding increased usage. In turn, to support these AI languages many new machines are being developed. These include workstations such as the SYMBOLICS 3600 [29] for LISP and ICOT PSI [32] for PROLOG, and also novel parallel "inference" machines.

Below we briefly examine these artificial intelligence programming languages and machine architectures. We start, however, with a brief overview of artificial intelligence.

ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) is the area of computer science concerned with the design of "intelligent" computer systems, namely systems exhibiting characteristics normally associated with human intelligence. The major AI areas of application are: knowledge-based expert systems - computer programs which embody the specialised "knowledge" of a human expert sufficient to perform as a consultant; natural language and speech processing - human-computer interaction by writing or speaking one of the "natural" languages like English, French or Japanese; robotics - the planning and performance of physical actions of a (mobile) robot to manipulate items and navigate successfully; theorem proving -

using deductions from hypotheses and intuitive skills to find a proof for a conjectured theorem in mathematics; automatic programming - systems that assist humans in some aspect of programming; combinatorial and scheduling problems - systems handling the problems concerned with specifying optimal schedules or combinations; and perception - understanding of complex input data, such as a scene, using knowledge about the things being observed.

The AI application area of knowledge-based expert systems is having perhaps the biggest influence on machines and languages for AI. A knowledge-based system program has a number of distinctive characteristics. Firstly, the program contains an explicit representation of empirical human knowledge. Secondly, this representation of knowledge is relatively easy to read and understand. Thirdly, the program is able to provide, like a human, an explanation of its reasoning. Lastly, the program is easily modified, as when additional knowledge is included.

Within the area of knowledge-based systems, the two central topics are firstly formalisms for knowledge representation and secondly methods for problem-solving and heuristic search. For knowledge representation three main formalisms are seeing extensive use: production rules, structured objects, and predicate logic. With production rules, the data-base of knowledge consists of rules in the form of condition-action pairs: "if this condition occurs, then do this action". With structured objects (e.g. semantic nets, frames) a net consists of "nodes" representing objects, concepts and events, and "links" between the nodes representing their interrelationships. With predicate logic knowledge is represented by formal logic allowing facts to be deduced using the rules of inference.

For problem-solving and heuristic search there are two main methods of formulating search problems, namely state-space and problem reduction. For each of these formulations there are a wide variety of methods for conducting the search. Problem-solving systems can usually be described in terms of three main components: a "database" - describing the task domain situation and goal; the "operators" - that are used to manipulate the database; and the "interpreter" implementing a control strategy for deciding what to do next. There are two main control strategies namely forward and backward reasoning. "Forward reasoning" works from an initial situation to one satisfying a goal condition. "Backward reasoning", in contrast, works from the goal statement by reducing it to one or more subgoals, which are in turn reduced, until solved.

In practice, most knowledge-based systems are based on "IF-THEN rules".

```
Rule R1  IF   condition_1 & condition_2 & ...
         THEN action_1 & action_2 & ...

Rule R2  IF        . . .
```

These rules may be used in two ways, either moving "forwards" from a condition to an action and so on, or moving "backwards" by hypothesising an action and then using the rules to verify the conditions.

Having briefly examined artificial intelligence (and knowledge-based systems its most important application area), we will next examine AI programming languages.

PROGRAMMING LANGUAGES

When choosing a language for AI programming a variety of options exist ranging from conventional languages such as PASCAL or FORTRAN, through languages such as LISP or PROLOG, to expert systems languages such as OPS5. Symbolic languages (e.g. LISP, PROLOG, OPS5) are usually chosen for work in AI for several reasons [11]. Firstly, these languages are oriented to the symbolic computation of AI, so their programs can easily manipulate symbols and their relationships. Secondly, these languages are frequently interactive, greatly facilitating the evolutionary development of AI software. Finally, the programmer is freed from organisational problems like memory management.

The choice of programming language, or even dialect, is frequently a matter of personal preference and availability, rather than obvious technical merits. In fact, languages from each of the major programming styles are being used for AI programming. As shown in Figure 2 there are at least five major categories of programming languages: Procedural, Object-Oriented, Functional, Logic and what we refer to as Application-Oriented programming.

Category	Examples
Procedural Programming	
conventional	FORTRAN, PASCAL
concurrent	
shared memory	ADA, MODULA-2
message passing	CSP, OCCAM
Object-Oriented Programming	SMALLTALK
Functional Programming	
applicative	
pattern-matching	LISP, KRC, HOPE
function-level	FP
data flow	ID, LUCID, VALID
Logic Programming	
Horn-clause	PROLOG
Application-Oriented Programming	
Expert-system	OPS5, ART

Figure 2: Categories of Programming Languages

Procedural programming is based on the concepts of the von Neumann control flow computer. These basic concepts are: a global memory of cells, assignment as the basic action, plus implicitly sequential control structures for the execution of statements. There are two subclasses of procedural languages, namely the conventional sequential languages (e.g. BASIC, FORTRAN, PASCAL) and concurrent languages. Concurrent languages extend the von Neumann model with parallel control structures based on processes, together with communication and synchronisation mechanisms. These concurrent languages may themselves be subdivided, by the way parallel processes communicate, into: shared memory (e.g. ADA) and message passing (e.g. OCCAM [24]).

Although procedural languages are the most familiar and are readily available on a range of machines, they are not really suitable for AI programming. They are generally not interactive, and provide only rudimentary facilities for symbol manipulation and memory management. However concurrent languages may find use where parallel problem-solving and heuristic search is being investigated.

Object-Oriented programming is based upon active objects, sometimes called actors, which communicate by passing messages. Every object belongs to a "class" and is created as an "instance" of that class. The class defines the detailed representation of its instances, the messages to which they can respond, and the "methods" for computing

the appropriate responses:

```
CLASS NAME cl

VAR v1, v2, ...

METHODS

  message_pattern_1 | local_vars_1 | body_1 .

  message_pattern_2 | local_vars_2 | body_2 .
 . . .
```

In an "instance" are stored the particular set of values that define its state. Each "method" has three parts: a message pattern which is similar to a label, some temporary variable names, and expressions to process the received message (these three parts of a method are separated by vertical bars "|").

The following example illustrates the distinction between Object-Oriented programming and Procedural programming:

> to evaluate <some object> + 4 means to present + 4 as a message to the object. The fundamental difference is that the object is in control, not the +. If <some object> is the integer 3 then the result will be the integer 7. However, if <some object> were the string META the result might be META4.

SMALLTALK [9] represents the current state of the art in object-oriented programming. An important aspect of SMALLTALK is that it provides a "total" programming system unifying features normally found in operating systems with those of programming languages. In addition, the object-oriented style of programming is becoming increasingly popular in AI, so SMALLTALK-like languages may find increased use.

Functional programming is based on a program being a "function" in the mathematical sense: it is applied to the inputs of the program and the resulting values are the program's output. These programs have a number of important properties. Firstly, a function's arguments and output values may be list structures, so a function can define complex operations on its inputs. Secondly, programs have "referential transparency" with the value of an expression depending only on its textual context and not on its computational history. Lastly, functional programs exclude assignment statements and side-effects.

Two important classes of functional programming languages can be identified: applicative languages and data flow (i.e. single-assignment) languages. In an applicative language (e.g. Pure LISP, KRC, FP) the basic concepts are [12]: the application of functions to structures and that all structures are expressions in the mathematical sense. In a data flow language (e.g. ID, LUCID, VAL, VALID) [1] the basic concepts are: that data "flows" from one statement to another, that execution of statements is data driven, and that identifiers obey the single-assignment rule.

However, the language most people think of when functional programming is mentioned is LISP [31]. However, LISP is strictly a functional language only if all functions with side-effects (e.g. REPLACA, REPLACD etc.) are avoided. This subset of the language is often referred to as Pure LISP [12]. LISP is the pre-eminent AI programming language with implementations like INTERLISP and MACLISP providing sophisticated programming environments.

Logic programming [16] is based on facts about a certain subject, stated as a collection of sentences, which can be used to solve problems or to answer questions. In a predicate logic language (e.g. PROLOG) [4,22] the basic concepts are: that statements are relations of a restricted form and that execution is a suitably controlled logical deduction from the statements.

The most widely known logic language, clearly, is PROLOG. In PROLOG statements are called "clauses" and the execution is a deduction from the clauses forming the program. The following "family tree" program consists of four clauses:

```
father(bill, john).
father(john, tom).

grandfather(X,Z):-  father(X,Y), mother(Y,Z).
grandfather(X,Z):-  father(X,Y), father(Y,Z).
```

Here the first two clauses define that "bill is the father of john", and "john is the father of tom". The second two clauses use the variables X,Y,Z to express the rule that X is the grandfather of Z, if X is the father of Y and Y is either the mother or father of Z. Such a program can be asked a range of questions, from "is john the father of tom" (expressed as father(john, tom)?) to "is there any A who is the grandfather of any C" (expressed as grandfather(A, C)?).

Usage of PROLOG in AI applications is expanding rapidly, largely as a result of PROLOG being adopted by the Japanese as the core language for their Fifth Generation project.

Expert System programming covers languages specifically for knowledge-based applications. These can be divided into three main groups, namely (i) skeleton systems called "Shells", (ii) languages based on "Rules", and (iii) languages based on a number of formalisms, which we refer to as "Multi-formalism".

Shells are basically expert systems without knowledge, for example EMYCIN derived from MYCIN and KAS derived from PROSPECTOR [11]. A Shell usually contains an applications-specific control strategy, and so any domain-specific knowledge must be represented as rules. Although Shells simplify the building process, they are not really programming languages. In addition, they are often limited in scope because the old structure may be an inappropriate framework for the new application.

Rule-based languages (e.g. OPS5, ROSIE) [11] are less constrained than Shells, having a general-purpose control strategy and memory operations. OPS5 [8], for example, consists of productions (i.e. rules) stored in the "production memory" that operate on expressions stored in a global data base called "working memory", under control of a simple forward-chaining "recognise-act" loop that scans the rules.

Multi-formalism languages integrate a number of different programming models (e.g. procedural, logic etc.), allowing a specific model to be used as appropriate. For instance LOOPS [19] provides: procedural programming, object-oriented programming, access-oriented (i.e. statements executed when data is accessed or changed) programming, and rule-based programming.

At the present time, for building expert system applications, Shells and Rule-based languages are finding increased usage. Shells provide an easy introduction to constructing expert systems, whereas Rule-based languages provide a powerful expert system framework without over-constraining the intended applications. In the future Multi-formalism languages, arguably still at the research stage, promise to provide sophisticated expert system tools.

MACHINE ARCHITECTURES

Currently the choice of machine for AI is largely dictated by the preference and availability of AI programming languages. The machines fall into three groups: personal computers (e.g. IBM PC), superminicomputers (e.g. VAX) and workstations (e.g. SYMBOLICS 3600 [29], Racal-Norsk KPS-10 [21]). However, only the LISP and PROLOG workstations are specifically designed for symbolic computation.

For a computer architecture to support symbolic computation the following mechanisms are appropriate:

- structured memory - a dynamic list-structure memory as built from LISP nodes plus garbage collection.

- tagging - associating a type tag with each item of data.

- control stacks - for holding the execution states of symbolic computations.

and for the future:

- processes - supporting multi-processing, together with communication and synchronisation between processes.

- parallelism - supporting concurrent execution of programs, including non-determinism.

As shown by Figure 1 above, there are at least six major categories of computer architecture that could potentially be used for AI: Control Flow, Data Flow, Reduction, Actor, Logic and Inference machines.

Control Flow computers embody the same computational model as procedural languages, namely: a vector of fixed-size memory cells, assignment as the basic action, and sequential execution of instructions. Computation in a control flow computer is characterised by explicit flow(s) of control causing the execution of instructions and data being passed between instructions via the shared memory cells.

Sequential control flow computers for symbolic computation centre on single-user workstations such as the SYMBOLICS 3600 [29] and Racal-Norsk KPS-10 [21] for LISP and the ICOT PSI [32] for PROLOG. These workstations are microprogrammed to support an abstract machine that operates a stack to evaluate programs. In the case of LISP this abstract machine is usually similar to the so-called SECD machine [12], whereas for PROLOG the abstract machine is usually the interpretation mechanism of the Edinburgh University DEC-10 PROLOG [32].

Future parallel control flow computers for symbolic computation are likely to centre on networks of new VLSI microcomputers, such as the TRANSPUTER [2]. The central idea of the TRANSPUTER is that each microcomputer is able to operate alone as a complete computer or as a component with other TRANSPUTERS in a parallel computer system. The TRANSPUTER (and its language OCCAM) is based on concurrent processes that communicate by sending messages through communication channels. A network of processes matches the target network of TRANSPUTERS, with each microcomputer supporting one or more processes. TRANSPUTER-like microcomputers provide a good base for parallel processing, but would need to be enhanced for symbolic computation by a structured

memory mechanism.

Data Flow computers embody the data flow or single-assignment programming model. The most important properties of data flow are that instructions pass their results directly to all the consuming instructions and that an instruction is executed when it has received all its inputs.

Data flow computers are usually based on a packet communication machine organisation [25]. This organisation consists of a circular instruction execution pipeline of resources in which processors, communications and memories are interspersed with "pools of work", as shown in Figure 3.

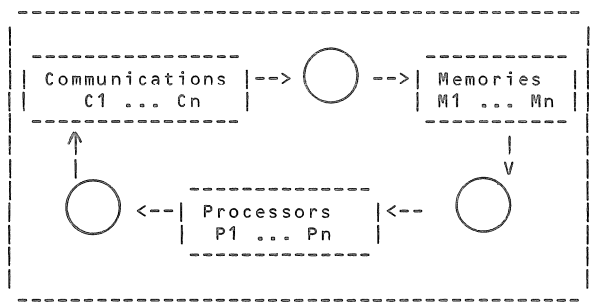


Figure 3: Packet Communication Computer

The organisation views an executing program as a number of independent information packets all of which are conceptually active, that split and merge. For a parallel computer, packet communication is a very simple strategy for allocating packets of work to resources. Each packet to be processed is placed with similar packets in one of the "pools of work". When a resource becomes idle it takes a packet from its input pool, processes it and places a modified packet in an output pool, returning then to the idle state.

A number of data flow machines are already operational [25], including the LAU system [23] in France and the Manchester Data Flow computer [10] in Britain. However, it is still unclear whether these data flow machines are suitable for symbolic computation.

Reduction computers embody the applicative model of functional programming where the basic concepts are [15,25]: the application of functions to structures and that all structures are expressions in the mathematical sense. In a reduction computer [25] the requirement for a result triggers the execution of the instruction that will generate the value, then control and the result are returned to the

invoking instruction. Recognition of reducible expressions and the transformation of these expressions forms the basis of execution. Execution is by a substitution process, which traverses the program structure and successively replaces reducible expressions by others that have the same meaning, until a constant expression representing the result of the program is reached.

A number of Reduction machines are under development [25,28]. One example is the ALICE parallel reduction machine [5] being built at Imperial College, London. ALICE has a packet communication organisation and is constructed from a number of TRANSPUTERS, acting as emulators. These TRANSPUTERS communicate via two rings, one of which carries packets of work and the other results. Since Reduction computers are designed to efficiently support applicative languages they should prove very effective for (Pure) LISP languages.

Actor computers embody the same model as object-oriented languages where the basic concepts are: objects are viewed as active, they may contain state, and objects communicate by sending messages. In an Actor computer the arrival of a message for an instruction causes the instruction to execute.

Computer architectures to support Object-Oriented languages are an area of growing interest. However, most of the projects involve the enhancement of Control Flow computers to support objects, rather than attempts to design an Actor computer embodying object-oriented concepts at the lowest (i.e. instruction) level. One proposal for a (parallel) Actor computer is that from Wilner [30].

Logic computers embody the predicate logic programming language where the basic concepts are: statements are relations of a restricted form, and execution is a suitably controlled logical deduction from the statements. In a logic computer an instruction is executed when it matches a target pattern and parallelism or backtracking is used to execute alternatives to the instruction.

Design of a Logic machine to execute PROLOG-like languages requires four essential mechanisms: unification, backtracking (or parallelism), tagging, and structured memory. Unification consists of initially searching for the called clause, next fetching the arguments of both the caller and callee predicates, and then examining the equality of the arguments. Backtracking (or parallelism) is needed if there are several clauses corresponding to the goal predicate and each must be examined to see if it makes the predicate true. Tagging of data is required for fast argument type checking and, lastly, a structured memory, supporting dynamic memory allocation, is required for fast

creation and deletion of variable size list structures.

Although there is considerable research in progress to design machines to support (parallel) PROLOG, the majority of these are not true Logic computers but are perhaps best classified as Control Flow, Data Flow or Reduction machines. An example being the ICOT PSI [32] based on Control Flow. One Logic computer project is Bibel's CONNECTION-METHOD machine [3], a multiprocessor architecture for deduction in first-order logic.

Inference computers is the name used here to define a class of parallel machines specifically embodying a knowledge-based formalism. Examples that might be included are BOLTZMANN machines [6,14], DADO [20] and Hillis' CONNECTION machine [13]. They have similarities to Bibel's CONNECTION-METHOD machine, but are based on alternative representations to predicate logic.

A BOLTZMANN machine, for instance, is based on a parallel constraint satisfaction network that is capable of learning the underlying constraints that characterise a domain. The machine is composed of primitive computing elements called "units" that are connected to each other by bi-directional "links". A unit is always in one of two states, "on" or "off", and it adopts these states as a probabilistic function of the states of its neighbouring units and the "weights" on its links to them. A unit being "on" or "off" is taken to mean that the system currently accepts or rejects some elemental hypothesis about the domain [6].

This class of massively parallel architectures marks the most novel end of the spectrum of research into machines for artificial intelligence. Although very speculative, any breakthrough would clearly have a major impact not only on machine architecture but more importantly on AI in general.

CONCLUSION

Over the last thirty years the focal-point of computing has moved from numeric (scientific) processing in the 50's, into data processing in the 60's and 70's, and on into symbolic processing in the 90's. This is being mirrored by the rapid expansion of programming languages and machine architectures for the support of Artificial Intelligence applications.

Regarding programming languages for AI, currently the choice is between languages like LISP and PROLOG, and Rule-based languages such as OPS5. (Shells can also be useful in specific domains). To build an AI application from scratch either LISP or PROLOG can be used. The actual choice between LISP or PROLOG is probably more a matter of personal preference, though LISP programming environments (e.g.

INTERLISP) are far more developed. However, to specifically build an expert system application Rule-based languages have advantages because they provide a framework for an expert system. For the future we can expect to see parallel symbolic languages, parallel variants of LISP and PROLOG, such as Concurrent PROLOG [18]. In addition, Multi-formalism languages may find increased usage.

Regarding machine architectures for AI, currently the debate appears to be between advocates of personal computers such as the IBM PC and of high-performance single-user workstations such as the SYMBOLICS 3600 or Racal-Norsk KPS-10. This choice is largely dictated by the proposed software to be used. For the future we can expect very high performance personal computers and workstations making use of parallel VLSI microcomputers such as the TRANSPUTER. In addition, parallel Logic machines and Inference machines, currently at the research stage, may make an impact on AI applications.

REFERENCES

[1] Ackerman W.B., "Data Flow Languages", IEEE COMPUTER, vol.15, no.2 (February 1982), pp. 15-25.

[2] Barron I. et al, "Transputer does 5 or more MIPS even when not used in parallel", Electronics, vol. 56, no. 23 (November 1983) pp. 109-115.

[3] Bibel W. and Buchberger B., "Towards a Connection Machine for Logical Inference", Techn. Universitat Munchen (1985).

[4] Clocksin W. and Mellish C., Programming in PROLOG, Springer-Verlag 1981.

[5] Darlington, J., & Reeve, M., "ALICE: A multiprocessor reduction machine for the parallel evaluation of applicative languages", in Proc. Int. Symp. Functional Programming Languages and Computer Architecture, (June 1981), pp. 32-62.

[6] Fahlman S.E., Hinton G.E. and Sejnowski T.J., "Massively Parallel Architectures for AI: NETL, Thistle and Boltzmann Machines", Proc. Nat. Conf. of AI AAAI-83 (August 1983) pp.109-113.

[7] Fredrikson E., "FGCS Introduction", Future Generation Computer Systems, vol. 1, no. 1 (July 1984) pp. 3-7.

[8] Forgy C.L., "The OPS5 user's Manual", Tech. Report CMU-CS-81-135, Computer Science Department, Carnegie-Mellon University (1981).

[9] Goldberg A. and Robson D., "SMALLTALK-80: The language and its implementation", Addison-Wesley (1983).

[10] Gurd J., Kirkham C. and Watson I., "The Manchester Data Flow Computer", CACM vol. 28, no. 1 (1985) pp. 34-52.

[11] Hayes-Roth F., et al., "Building Expert Systems", Addison-Wesley (1983).

[12] Henderson P., "Functional programming: Application and Implementation", Prentice-Hall (1980).

[13] Hillis W.D., "The Connection Machine", Tech. Report 646, MIT, AI Lab. (1981).

[14] Hinton G.F., Sejnowski T.J. and Ackley D.H., "Boltzmann Machines: Constraint Satisfaction Networks that Learn", Tech. Report CMU-CS-84-119, Carnegie-Mellon University (May 1984).

[15] Kluge W.E., "Cooperating Reduction Machines", IEEE Trans. on Computers, vol. C-32, no. 11 (Nov. 1983) pp. 1002-

[16] Kowalski R., "Logic for Problem Solving", North-Holland (1979).

[17] Moto-oka T., "Overview to the fifth generation computer system project", Proc. Tenth Int. Symp. on Computer Architecture (June 1983).

[18] Shapiro E.Y., "A Subset of Concurrent Prolog and its Interpreter", Tech. Rep. of ICOT, TR-003 (February 1983).

[19] Stefik M. et al, "Knowledge Programming in Loops: report on an experimental course", The AI Magazine, vol. 4, no. 3 (Fall 1983) pp. 3-13.

[20] Stoflo S.J. Shaw D.E., "DADO: A Tree-Structured Machine Architecture for Production Systems", Proc. Nat. Conf. on AI AAAI-82 (1982).

[21] Racal-Norsk., "Preliminary Information on the Racal-Norsk Knowledge Processing Systems", (1984)

[22] Rousell P., "PROLOG Mannel de reference et d'utilisation", Groupe d'Intelligence Artificielle, Marseille - Luming (1975)

[23] Syre J-C., Comte D., and Hifdi N., "Pipelining, parallelism and asynchronism in the LAU System", Proc. Int. Conf. Parallel Processing (1977) pp. 87-92

[24] Taylor R. and Wilson P.: "OCCAM Process-oriented language meets demands of distributed processing", Electronics (November 1982) pp. 89-95.

[25] Treleaven P.C. et al: "Data Driven and Demand Driven Computer Architecture", ACM Computing Surveys, vol. 14, no. 1 (March 1982) pp. 93-143.

[26] Treleaven P.C. and Gouveia Lima I., "Japan's Fifth Generation Computer Systems", IEEE COMPUTER, vol. 15, no. 8 (August 1982) pp. 79-88.

[27] Treleaven P.C. and Gouveia Lima I., "Future Computers: logic, data flow,...,control flow?", IEEE COMPUTER, vol. 17, no. 3 (March 1984) pp. 47-57.

[28] Vegdahl S.R., "A Survey of Proposed Architectures for Execution of Functional Languages", IEEE Trans on Computers, vol. C-33, no. 12 (Dec. 1984) pp. 1050-1071.

[29] Weinreb D. and Moon D., "Lisp Machine Manual", Symbolics Inc.

[30] Wilner W.T., "Recursive Machines", Xerox Palo Alto Research Center, Internal Report (1980).

[31] Winston P. and Horn B., "LISP", Addison-Wesley, Reading 1981.

[32] Yokota M. et al, "The Design and Implementation of a Personal Sequential Inference Machine: PSI", New Generation Computing, vol. 1, no. 2 (1983).

V CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
XI CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA

COMISSÃO ORGANIZADORA

PRESIDENTE DO CONGRESSO: Claudio Zamitti Mammana

COMISSÃO PROMOTORA:

Prof. Francisco Luis dos Santos Ferraz - **Reitor da UFRGS**
Prof. Claudio Zamitti Mammana - **Presidente da SBC**
Prof. Jorge Vidart - **Presidente do CLEI**

COORDENAÇÃO GERAL:

Manoel Agamemnon Lopes - **SBC**
Aldo Migliario Osório - **CLEI**

COMISSÃO EXECUTIVA:

Coordenador:

Simão Sirineo Toscani	UFRGS
Carlos Ernesto Rech	UFRGS
Cláudia Sabani	UFRGS
Clésio Saraiva dos Santos	UFRGS
Helena Vauthier de Souza	UFRGS
Luiz Julião Braga Filho	UFViçosa
Paulo R.F. Mosca	UFRGS
Philippe O.A. Navaux	UFRGS
Ricardo A.L. Reis	UFRGS
Roberto M.J. Macedo	UFRGS
Sergio Felipe Zirbes	UFRGS
Therezinha da Costa Chaves	UFRJ

SECRETARIA EXECUTIVA:

Lourdes Tassinari	UFRGS

COMISSÃO DO SEMISH:

Coordenação Geral:

Ricardo Augusto da Luz Reis	UFRGS

Comissão Nacional:

Coordenador:

Ricardo Augusto da Luz Reis	UFRGS
Antonio Carlos R. Costa	UFRGS
Artur João Catto	UFSCar
Leo Pini Magalhães	UNICAMP
Merval Jurema Filho	UFPe
Michael Stanton	PUC/RJ
Newton Faller	UFRJ
Roberto da Silva Bigonha	UFMG
Siang Wun Song	USP

Comissão Latino-Americana
Coordenador
Roberto M.J. Macedo
Universidade Federal do Rio Grande do SUL - **Brasil**

Hector Rodrigues Estay
Universidade Católica de Valparaiso - **Chile**

Juan M. Pereira
Universidad Simón Bolivar - **Venezuela**

Leonardo Elizalde
Pontifícia Univ. Católica de Equador - **Equador**

Leopoldo Carranza
Sadio - **Argentina**

Oscar Herreros
Universidad Católica de Assunción - **Paraguai**

Simon M. Tenzer
Universidad de la República - **Uruguai**

Tulio Galvez
Associación Peruana de Computación e Informática - **Peru**

Victor Toro
Universidad de los Andes - **Colombia**

Grigot Ponce de Leon
Universidade Privada de Santa Cruz - **Bolivia**

COMISSÃO DO SBCCI

Coordenador:

Ricardo Augusto da Luz Reis	CPGCC/UFRGS
Altamiro Amadeu Suzim	CPGCC/UFRGS
Ana Maria Kuniyoshi	IM/CTI
Antonio Carlos Barbosa	ITAUCOMP
Carlos Ignácio Mammana	IM/CTI
Eber Assis Schmitt	NCE/UFRJ
Jean Albert Bodinaud	IME/USP
Mario Vaz da Silva	DEE/UFRJ
Victor Blatt	CPqD/TELEBRÁS

COMISSÃO DO SECOMU

Coordenador

Henrique Paca Loureiro Luna	UFMG

COMISSÃO DO JAI

Coordenadora:

Lidia Michaela Segre	COPPE/UFRJ

COMISSÃO DO CTIC

Coordenadora:	
Alaide Mammana	UNICAMP
Paulo Alberto de Azeredo	UFRGS
Lilian Markenson	COPPE/UFRJ
Mario Ferraretto	CTI
Newton Faller	UFRJ

COMISSÃO DA EXTEC

Coordenador:	
Tiaraju Wagner	UFRGS
Carlos Ernesto Rech	UFRGS
Rejane Telichevesky	UFRGS

COMISSÃO CULTURAL

Coordenadora:	
Cláudia Sabani	UFRGS
Helena Vauthier de Souza	UFRGS
Cleto José Beuren	UFRGS

COMISSÃO DE EDIÇÃO

Coordenador:	
Carlos Ernesto Rech	UFRGS
Nicolau Meisel	PUC/RJ

SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

Presidente: Cláudio Zamitti Mammana	USP
Vice-Presidente: Clésio Saraiva dos Santos	UFRGS
Secretário Geral: Manoel Agamenon Lopes	UFPe
1º Secretário: Oscar Luis Monteiro de Farias	BNDES
2º Secretário: Mário Ferrareto	CTI
Tesoureiro: Arlindo Vasquez Martins	COBRA

Conselho:

Arndt von Staa	PUC/RJ
Daniel Menascé	PUC/RJ
Estevam Gilberto de Simone	UFRJ
Ivan Moura Campos	UFMG
Lidia Michaela Segre	COPPE/UFRJ
Luis de Castro Martins	PUC/RJ
Luiz Julião Braga Filho	UFViçosa
Roberto da Silva Bigonha	UFMG
Simão Sirineo Toscani	UFRGS
Sueli Mendes dos Santos	UFRJ

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. Francisco Luis dos Santos Ferraz

Vice-Reitor: Prof. Gerhard Jacob

Pró-Reitor de Planejamento: Prof. Roberto Alves Pinto

Pró-Reitor de Administração: Prof. Luis Carlos Ribeiro Bortolini

Pró-Reitor de Extensão: Prof: Flávio Giannetti Loureiro Chaves

Pró-Reitor de Graduação: Prof. Walter Otto Cybis

Pró-Reitor de Assistência à Comunidade Universitária:
Prof. Valentim Emílio Uberti Costa

Pró-Reitor de Pesquisa e Pós-Graduação
Prof. Hélgio Henrique C. Trindade

CENTRO LATINO-AMERICANO DE ESTUDOS EM INFORMÁTICA

Presidente:
Jorge Vidart
Universidade Simón Bolivar - Caracas- **Venezuela**

Secretário Executivo:
Aldo Migliario Osorio
Universidade Católica de Valparaiso - **CHILE**

Comitê Diretor:
Daniel E. Gascue
Centro Interdisciplinar de Estudos de Desenvolvimento do Uruguai-
Montevideo- **Uruguai**

Newton Braga Rosa - Universidade Federal do Rio Grande do Sul - **Brasil**

Carla Bruschi de Cardinales
Universidad Nacional de San Juan- San Juan- **Argentina**

Corrado Vallet
Universidade Gabriel Rene Moreno - Santa Cruz- **Bolivia**

Wilfredo Kleeberg H.
Associación Peruana de Computación e Informática- Lima- **Peru**

Germán Hernan
Associación Colombiana de Cálculo Eletrônico e Investigación Operativa-
Bogotá- **Colombia**

Luis Alberto Meyer
Faculdad de Ciências e Tecnologia- Universidad Católica de Assunción-
Assuncíon- **Paraguai**

Carlos Ciuffardu Pace
Universidad Catóolica de Valparaiso- **CHILE**

Diego Andrade Stacey
Universidad Católica de Equador - Quito - **Equador**