XIII CONFERENCIA LATINOAMERICANA DE INFORMATICA

XI CONGRESO COLOMBIANO DE CALCULO ELECTRONICO E INVESTIGACION OPERACIONAL

ORGANIZADO POR:

ACCIO
ASOCIACION COLOMBIANA DE INFORMATICA

UNIVERSIDAD DE LOS ANDES. BOGOTA

AUSPICIADO POR:

CLEI
CENTRO LATINOAMERICANO DE ESTUDIOS EN INFORMATICA

Noviembre 9 al 13 de 1987
Bogotá, Colombia

EDGAR ENRIQUE RIVAS ORTIZ
OSCAR LOPEZ BELTRAN

A.O
25/87
I

XIII CONFERENCIA LATINOAMERICANA DE INFORMATICA

XI CONGRESO COLOMBIANO DE CALCULO ELECTRONICO E INVESTIGACION OPERACIONAL

Bogotá, 9 al 13 de Noviembre de 1987

ORGANIZADO POR:

CLEI
Centro Latinoamericano de Estudios en Informática

ACCIO
Asociación Colombiana de Informática

UNIANDES
Universidad de los Andes - Bogotá

AUSPICIADO POR:

UNESCO
Oficina Regional de Ciencia y Tecnología
para América Latina y el Caribe

CON LA COLABORACION DE:

- Universidad Nacional de Colombia
- Universidad Piloto de Colombia
- Universidad Autónoma de Manizales

Los organizadores de la "XIII COFERENCIA LATINOAMERICANA DE INFORMATICA" y del "XI CONGRESO COLOMBIANO DE CALCULO ELECTRONICO E INVESTIGACION OPERACIONAL" agradecen muy especialmente la colaboración y apoyo que han brindado las siguientes compañías e instituciones:

El Tiempo
El Heraldo
Avianca
ICM de Colombia Ltda.
Teorema Ltda.
Lógica Ltda.
Procálculo S.A.
Gémini Computadores
UNISYS de Colombia
CopiTécnica Ltda.
Mac Graw Hill
El Espectador
El Siglo
Aviatur S.A.
Coldatos S.A.
Xerox de Colombia S.A.
Colinversiones S.A.
J. Glottmann S.A.
Ecopetrol
SuperMicros y Sistemas S.A.
Diners Club de Colombia S.A.
Colpuertos
La República
El Colombiano
Texins de Colombia S.A.
Educar Informática
Apple Computer Inc.
NCR de Colombia S.A.
MicroMundo
IBM de Colombia S.A.
Legis Ltda.
Ceina S.C.
Telecom

Agencia de Cargas y Turismo de Carga y Aduanas
Compañia Colombiana de Computadores Colcomp S.A.
Empresa Nacional de Computadores y Sistemas S.A.
Sistemas de Tecnología Avanzada S. T. A.
Secretaría de Informática de la Presidencia de la República
Data Processing Supplies Ltda.
Financiera Eléctrica Nacional S.A.
Centro Distrital de Sistematización y Servicios Técnicos SISE
Instituto Nacional de Investigaciones Geológicas y Mineras IngeoMinas
Fondo Colombiano de Investigaciones Científicas COLCIENCIAS

EXORDIO

El espíritu de los tiempos y el estado del arte de la tecnología obligan a considerar a la informática como un factor fundamental en el desenvolvimiento político, económico, social y cultural de cualquier nación.

El impacto económico de la informática, habida cuenta de la amplitud del ámbito que cubre y de la profundidad del mismo, es de tales proporciones que las consecuencias de ignorar el fenómeno, el tratarlo parcialmente como una tecnología más a utilizar, o el no dedicarle con la debida oportunidad el esfuerzo a encauzarlo en bien del progreso nacional, puede provocar un retraso quizás irremediable.

En este orden de ideas resulta inquietante la cuestión sobre el papel que pueden jugar nuestras débiles economías latinoamericanas en el contexto del fenómeno informático mundial, en el cual, en gran medida, sólo nos ha correspondido ser receptores pasivos de tecnologías provenientes de los países desarrollados que concentran en proporción aplastante el poder en este determinante cambio.

Es entonces cuando cobra sentido impulsar un proceso integracionista latinoamericano en materia de informática, por cuanto es apenas obvio deducir las ventajas que se desprenden de la cooperación e integración entre los países de América Latina, promovidas en el marco de una política global, frente a la alternativa de acometer la empresa aisladamente, con la consiguiente duplicación de esfuerzos, debilidad negociadora y pérdida de la capacidad de alcance de prometedores efectos sinergéticos.

Pero ¿ cómo "hacer el camino" en una región donde los intentos integracionistas se han distinguido, si no por su total fracaso, tampoco por su estruendoso éxito ?.

La respuesta parece encontrarse en el dominio de las aproximaciones sucesivas. Los acuerdos suscritos a nivel de los gobiernos, si bien son necesarios, en absoluto resultan suficientes. La crisis a que se vió abocado recientemente el Pacto Subregional Andino, es un claro ejemplo al respecto. Es indispensable complementariamente propiciar el acercamiento entre los estamentos docentes, de investigación y empresariales de los países que integran la región. Reforzar las relaciones que presenta el MODELO SABATINO, compartir el "savoir-faire" alcanzado, propender por la apropiación de tecnología para la región, establecer convenios en los campos de la educación y la investigación, y consorcios en el de la industria - que de por sí estaría aprovechando las ventajas que brinda el mercado ampliado -, y en fin, fomentar el intercambio de inquietudes, son, todos, objetivos deseables, y son precisamente los que han alentado la realización de las sucesivas versiones de la CONFERENCIA LATINOAMERICANA DE INFORMATICA, evento enhorabuena creado por el Centro Latinoamericano de Estudios en Informática CLEI, que ha venido deambulando por la amplia geografía de la gran patria latinoamericana con la plena certeza de la altísima misión que se ha impuesto.

El desafío es grande, pero parece el único camino viable, si es que los países de América Latína, que es nuestra América, desean encontrar el sendero que los conduzca al anhelado desarrollo socio-económico de manera sostenida y con una mayor autonomía, así como lograr una inserción más equitativa en la economía internacional y una digna reafirmación de sus irrenunciables valores culturales en el concierto de las naciones del universo-mundo.

En esta oportunidad, gracias al esfuerzo mancomunado de la ASOCIACION COLOMBIANA DE INFORMATICA-ACCIO y de la UNIVERSIDAD DE LOS ANDES, así como de otras instituciones que se han sumado al empeño, Colombia abre sus puertas generosamente a los hermanos latinoamericanos, y al evocar con su nombre la magnífica osadía del Gran Almirante, es del caso asímismo, proclamar la invitación a atrevernos a llevar a la práctica, al menos en el campo informático, la idea formidable que en el desafortunadamente fallido Congreso Anfictiónico, animó al más grande visionario de América: Simón Bolivar, cuyo legado tenemos el ineludible compromiso de acrecentar.

Jaime Obregón Puyana
Predidente
Asociación Colombiana de Informática ACCIO

Arturo Infante Villarreal
Rector
Universidad de los Andes

Noviembre de 1987

COMITE DE HONOR

XIII CONFERENCIA LATINOAMERICANA DE INFORMATICA
XI CONGRESO COLOMBIANO DE CALCULO ELECTRONICO E INVESTIGACION OPERACIONAL

Dr. Virgilio Barco V.
Presidente de la República

Dr. Fernando Cepeda U.
Ministro de Comunicaciones

Dr. Fuad Char A.
Ministro de Desarrollo Económico

Dr. Pedro Amaya P.
Director del Fondo Colombiano de Investigaciones Científicas COLCIENCIAS

Dr. Alfonso González C.
Director del Departamento Administrativo Nacional de Estadística DANE

Dr. Emilio Aljure N.
Director del Instituto Colombiano para el Fomento de la Educación Superior ICFES

Dr. Ernesto LLeras M.
Presidente de la Asociación de Ingenieros de Sistemas ACIS

Dr. Juan José González
Presidente de la Asociación Colombiana de Usuarios de Computador ACUC

Dr. Alberto Restrepo V.
Presidente de la Asociación de Usuarios de Computador AUC

Dr. Jorge Vidart
Presidente del Centro Latinoamericano de Estudios en Informática CLEI

Dr. Aldo Migliaro O.
Secretario Ejecutivo del Centro Latino-Americano de Estudios en Informática CLEI

PRESIDENCIA COLEGIADA

XIII CONFERENCIA LATINOAMERICANA DE INFORMATICA
XI CONGRESO COLOMBIANO DE CALCULO ELECTRONICO E INVESTIGACION OPERACIONAL

Arturo Infante Villarreal
Rector
Universidad de los Andes

Jaime Obregón Puyana
Presidente Asociación
Colombiana de Informática ACCIO

CENTRO LATINOAMERICANO DE ESTUDIOS EN INFORMATICA

CLEI

CONSEJO DIRECTIVO

Presidente

Jorge Vidart
Escuela Latinoamericana de Informática ESLAI
Argentina

Secretario Ejecutivo

Aldo Migliaro Osorio
Universidad Católica de Valparaíso
Chile

Victor Manuel Toro C. [posesión en Noviembre'87]
Universidad de los Andes
Colombia

Miembros

Newton Braga Rosa
Universidad Federal de Rio Grande do Sul
Brasil

Jaime Obregón Puyana
Asociación Colombiana de Informática ACCIO
Colombia

José Pino Urtubia
Universidad de Chile
Chile

Luis Alberto Meyer
Universidad Católica del Paraguay
Paraguay

Daniel Gascue [tesorero CLEI]
Centro Interdisciplinario de Estudios CIEDUR
Uruguay

Julián Araoz
Universidad Simón Bolivar
Venezuela

COMITE DE PROGRAMA

Presidente

Rodrigo López B.
Universidad de los Andes
Colombia

Miembros

Marcos Borges
Universidad Federal de Rio de Janeiro
Brasil

José Durán
Universidad de Santa María
Chile

Victor Gamarra Herrera
Universidad Nacional de Ingeniería
Perú

Fernando Brum
Interfase Ltda.
Uruguay

Alejandro Teruel
Universidad Simón Bolivar
Venezuela

COMITE EJECUTIVO
XIII CONFERENCIA LATINOAMERICANA DE INFORMATICA
XI CONGRESO COLOMBIANO DE CALCULO ELECTRONICO E INVESTIGACION OPERACIONAL

Presidente

Victor Yockteng M.
Asociación Colombiana de Informática - ACCIO

Miembros

Rodrigo López B.
Universidad de los Andes

Victor Manuel Toro C.
Universidad de los Andes

Ricardo Amaya Umaña
Asociación Colombiana de Informática ACCIO

Pilar Murcia
Universidad de los Andes

Rubby Casallas G.
Universidad de los Andes

Secretaria Ejecutiva

María Isabel Mier F.

Asistente de Contabilidad

Fabio Orozco

=====*=====

COMITE ACADEMICO
XIII CONFERENCIA LATINOAMERICANA DE INFORMATICA
XI CONGRESO COLOMBIANO DE CALCULO ELECTRONICO E INVESTIGACION OPERACIONAL

Presidente

Rodrigo López B.
Lenguajes de Programación - UniAndes

Miembros

José Abásolo
Bases de Datos-UniAndes

Rodrigo Cardoso
Informática Teórica-UniAndes

Carlos Cortés
Teoría de Sistemas-U. Nacional

Alfonso Charún
Telecomunicaciones-U. Nacional

Mauro Flórez
Telecomunicaciones-U.Piloto

Alvaro Galvis P.
Educ. Apoyada por Comp.-UniAndes

Jairo Gómez
Bases de Datos-B.C.H.

José Tiberio Hernández
CAD/CAM-UniAndes

Camilo Rueda
Inteligencia Artificial-U.A.Manizales

Victor Manuel Toro C.
Ingeniería de Software-UniAndes

ASESORES ACADEMICOS
XIII CONFERENCIA LATINOAMERICANA DE INFORMATICA
XI CONGRESO COLOMBIANO DE CALCULO ELECTRONICO E INVESTIGACION OPERACIONAL

Jaime Bohórquez
Programación Lógica-UniAndes

Juan Manuel Damián
Arq. Computadores-UniAndes

María Consuelo Franky
Sistemas Distribuídos-UniAndes

Olga Lucía Giraldo
Estruc. Bases Datos-UniAndes

Roberto Ojeda
Leng. Programación-U.Nacional

Mario José Otero
Paralelismo-UniAndes

Ignacio Pérez V.
Inv. Operacional-UniAndes

Alejandro Quintero
Sist. Operacionales-UniAndes

Leonardo Ramírez
Sistemas Públicos-UniAndes

Francisco Rueda F.
Sist. Operacionales-UniAndes

Jorge Villalobos
Interf. Hombre-Máquina - UniAndes

==*==

ASOCIACION COLOMBIANA DE INFORMATICA
ACCIO

Presidente:	Jaime Obregón Puyana	Ricardo Amaya Umaña
VicePresidente:	Victor R. Yockteng M.	Héctor Múnera Orozco
Secretario:	José Arias Ordóñez	Javier Francisco Garcés
Tesorero:	Alvaro Varela Flórez	Rómulo Niño Neissa
Vocales:	Carlos Cortés Amador	Mauro Flórez Calderón
	Carlos Ardila Arenas	María Esther Ordóñez O.
	Manuel Neira Rojas	Alvaro Zarama Medina
Revisor Fiscal:	Armando García H.	Ana María Pacheco
Directora Ejecutiva:	María Isabel Mier	

==*==

UNIVERSIDAD DE LOS ANDES
UNIANDES

Arturo Infante Villarreal
Rector

Ernesto Gühl Nannetti
ViceRector

Antonio García Rozo
Decano Facultad de Ingeniería

Francisco Rueda Fajardo
ViceDecano Facultad de Ingeniería

Ernesto LLeras Manrique
Director Centro de Investigaciones de Ingeniería

Rodrigo A. López B.
Director Dpto. Sistemas y Computación

INDICE

VOLUMEN I

INFORMATICA TEORICA E INFORMATICA SIMBOLICA

META-CONTROL Y PROGRAMACION LOGICA
ADOLFO KVITCA
ESCUELA SUPERIOR LATINOAMERICANA DE INFORMATICA - ESLAI - ARGENTINA 1

BACKTRACKING INTELIGENTE PARA PROLOG
SILVIA TAKAHASHI
UNIVERSIDAD DE LOS ANDES - COLOMBIA 17

PROGRAMACION LOGICA BASADA EN ASERCIONES: UNA HEREJIA IMPERATIVA.
JAIME BOHORQUEZ V.
UNIVERSIDAD DE LOS ANDES - COLOMBIA 27

UNA EXTENSION DE LISP PARA LA PROGRAMACION ORIENTADA A OBJETOS
GUSTAVO ARANGO
UNIVERSIDAD DE LOS ANDES - COLOMBIA 54

LOGICAS NO MONOTONICAS
RAUL CARNOTA
ESCUELA SUPERIOR LATINOAMERICANA DE INFORMATICA - ESLAI - ARGENTINA 67

IGUALDAD DEFINICIONAL DE EXPRESIONES MATEMATICAS EN LA LOGICA INTUITIVA
JUAN JOSE CABEZAS
UNIVERSIDAD DE LA REPUBLICA - URUGUAY 89

LA IGUALDAD EN TIPOS ABSTRACTOS DE DATOS
RODRIGO CARDOSO R.
UNIVERSIDAD DE LOS ANDES - COLOMBIA 110

PROCEDIMIENTO DE DECISION PARA LA EQUIVALENCIA Y LA CONGRUENCIA OBSERVACIONAL DE TERMINOS REGULARES DE CCS
AMELIA SORIANO MONTES
UNIVERSIDAD CENTRAL DE VENEZUELA - VENEZUELA 118

EDITOR DE AUTOMATAS FINITOS
FELIPE S. QUADROS; ANA M. PRICE
UNIVERSIDADE FEDERALE DO RIO GRANDE DO SUL - BRASIL 138

LOS CONJUNTOS DIFUSOS EN LA MODELIZACION LINGUISTICA
FRANCESCA GRESELIN FERRARI
UNIVERSIDAD CATOLICA DE NUESTRA SENORA DE LA ASUNCION - PARAGUAY 150

EL DICCIONARIO ELECTRONICO DEL ESPAÑOL
CARLOS SUBIRATS RUGGEBERG
UNIVERSIDAD AUTONOMA DE BARCELONA - ESPAÑA 160

ON THE CONNECTIONS BETWEEN RELATIONAL TABLEAUX AND CLAUSE FORM
PEDRO MANUEL SILVEIRA
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - BRASIL 173

ON A MIN-MAX CONJECTURE FOR REDUCIBLE DIGRAPHS
JAYME LUIZ SZWARCFITER
UNIVERSIDADE FEDERALE DO RIO DE JANEIRO - BRASIL 181

RESOLUÇAO DE EQUAÇOES: UMA METODOLOGIA DE ESTUDO
TIARAJU ASMUZ DIVERIO
INSTITUTO DE INFORMATICA PUC RS - BRASIL 190

UNA VARIANTE DEL SISTEMA DE ROTORES
ERNESTO AZORIN; NELSON BALOIAN
UNIVERSIDAD DE CHILE - CHILE 204

PROPOSICION DE UN LENGUAJE DE NIVEL INTERMEDIO PARA UNSISTEMA DE PRODUCCION DE COMPILADORES
A. CAMPOS; Y. ETEROVIC; G. GARCIA; R. SANTIS
UNIVERSIDAD CATOLICA DE CHILE - CHILE 215

BASES PARA UN LENGUAJE DE DESCRIPCION DE LA SEMANTICA DE LOS LENGUAJES DE PROGRAMACION
A. CAMPOS; Y. ETEROVIC; G. GARCIA; R. SANTIS
UNIVERSIDAD CATOLICA DE CHILE - CHILE 225

DESARROLLO E IMPLEMENTACION DE UN INTERPRETADOR DE LENGUAJE PROLOG PORTABLE Y EXTENDIBLE
LUIS ROBERTO OJEDA; CAMILO PARRA
UNIVERSIDAD NACIONAL DE COLOMBIA - COLOMBIA 236

INGENIERIA DE SOFTWARE Y TECNICAS DE PROGRAMACION

VERIFICAÇAO E VALIDAÇAO DE SOFTWARE NA FASE DE ESPECIFICAÇAO DE REQUISITOS
CLIFTON EDUARDO CLUNIE BEAUFONT; ANA REGINA CALVANCANTI DE ROCHA
UNIVERSIDAD TECNOLOGICA DE PANAMA - PANAMA 237

PARADIGMAS PARA O PROCESSO DE PRODUÇAO DE SOFTWARE E SUAS CONSEQUENCIAS SOBRE OS AMBIENTES DE DESENVOLVIMENTO
SUELI MENDES; ANA REGINA ROCHA
UNIVERSIDADE FEDERALE DO RIO DE JANEIRO - BRASIL 257

DOS APLICACIONES DE LAS TABLAS DE DECISION
LUIS ROBERTO OJEDA CH.
UNIVERSIDAD NACIONAL DE COLOMBIA - COLOMBIA 272

UMA ANALISE REALISTA DA UTILIZAÇAO DO ENFOQUE DA PROTOTIPAGEM RAPIDA NO DESENVOLVIMIENTO DE SISTEMAS DE INFORMAÇAO
MARCOS R. S. BORGES
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - BRASIL 280

EITIS: UM AMBIENTE DE FERRAMENTAS INTEGRADAS PARA DESENVOLVIMENTO DE SISTEMAS INTERACTIVOS
RUBENS N. MELO
PUC-RJ - BRASIL 290

DISEÑO ESPECIFICACION Y PROTOTIPIFICACION DE SISTEMAS INTERACTIVOS
VICTOR M. TORO C
UNIVERSIDAD DE LOS ANDES - COLOMBIA 302

CARTELERA DINAMICA : DISENO E IMPLEMENTACION
GERMAN RIVERA; IVAN H. REYES
UNIVERSIDAD DE LOS ANDES - COLOMBIA 326

ARQUITECTURA DE UNA INTERFASE USUARIO: UN MODELO GENERAL.
JORGE VILLALOBOS S.
UNIVERSIDAD DE LOS ANDES - COLOMBIA 333

CONSTRUCCION AUTOMATIZADA DE TALLERES DE PROGRAMACION: PEPIM
MIGUEL SANTANA
PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU - PERU 348

MESTRA: UMA METODOLOGIA DE ESCRITA DE DOCUMENTAÇAO INTERNA TRANSPORTAVEL
CAMILO DE LELIS GONDIM MEDEIROS
UNIVERSIDADE FEDERAL DA PARAIBA - BRASIL 363

INSTRUMENTADOR DE PROGRAMAS 'C'
MARCELO S. PIMENTA; VERNER HEIDRICH; ANA M. A. PRICE
UNIVERSIDADE FEDERALE DO RIO GRANDE DO SUL - BRASIL 376

SALTOS NO LOCALES: UNA EXTENSION A TURBO-PASCAL
RODRIGO LOPEZ B.
UNIVERSIDAD DE LOS ANDES - COLOMBIA 388

A PROBLEMATICA DO DESENVOLVIMIENTO DE SOFTWARE NUMERICO
TIARAJU ASMUZ DIVERIO
INSTITUTO DE INFORMATICA PUC RS - BRASIL 398

EL LENGUAJE DECLARATIVO COMO HERRAMIENTA DE IMPLEMENTACION
JUAN GROMPONE; F. BRUM; M. CASAMAYOR; F. SIMINI
INTERFASE LTDA. - URUGUAY 415

PROGRAMAÇAO LOGICA E ENGENHARIA DE SOFTWARE
TERESA CRISTINA DE AGUILAR
UNIVERSIDADE FEDERALE DO RIO DE JANEIRO - BRASIL 427

INGENIERIA DE SOFTWARE Y PROLOG
F. JAVIER DIAZ
ESCUELA SUPERIOR LATINOAMERICANA DE INFORMATICA - ESLAI - ARGENTINA 436

IMPLANTACION DE UN SISTEMA DE CONTROL INTERNO EN UN CENTRO DE PROCESAMIENTO DE DATOS
CARLOS A. PETRELLA
MINISTERIO DE TRASPORTE Y OBRAS PUBLICAS - URUGUAY 447

PROTESTE: UMA FERRAMENTA DE APOIO AO TESTE DE PROGRAMAS
ANA M. PRICE
UNIVERSIDADE FEDERALE DO RIO GRANDE DO SUL - BRASIL 461

LAS REDES DE PETRI COMO UN INSTRUMENTO PARA MODELAR DIAGRAMASDEFLUJOS DE DATOS JERARQUICOS
RICARDO ACEVEDO
UNIVERSIDAD SANTA MARIA - CHILE 463

METODOLOGIAS INTEGRADAS DE SISTEMATIZACION
JORGE ELIECER BARRERA NIÑO
BAVARIA - COLOMBIA 464

COMPUTACION GRAFICA Y DISEÑO ASISTIDO POR COMPUTADOR

MGIT-ANDES: UN MODELADOR 3-D SOBRE MICROCOMPUTADORES BASADO EN OPERADORES DE EULER
JOSE TIBERIO HERNANDEZ P.; JUAN C. QUIROGA CH.
UNIVERSIDAD DE LOS ANDES - COLOMBIA 466

USO DE HEURISTICOS EN COMPILACION DE UN LENGUAJE DE ALTO NIVEL PARA MAQUINADO EXTERNO EN TORNOS DE CONTROL NUMERICO
OSCAR RUIZ S.; RODRIGO LOPEZ B.
UNIVERSIDAD DE LOS ANDES - COLOMBIA 480

FLUX2D Y FLUX3D SOFTWARES DE CAD PARA LA MODELIZACION DE CAMPOS ELECTROMAGNETICOS
ORIETTA SANTANA; GERARD MEUNIER; JEAN LOUIS COULOMB; JEAN CLAUDE SABONNADIERE
ENSIEG - FRANCIA 501

COMPUTACION GRAFICA: METODOS DE APROXIMACION DE CURVAS Y SUPERFICIES
DANIEL FORMICA; CLAUDIO DELRIEUX; DANIEL MAGNI
DEPARTAMENTO DE INGENIERIA ELECTRICA - ARGENTINA 514

INTRODUCCION A LA TOMOGRAFIA AXIAL COMPUTARIZADA
JOSE J. RAMOS RODRIGUEZ
ECUADOR 532

SISTOM: UN SISTEMA PARA VISUALIZAÇAO TRIDIMENSIONAL DE TOMOGRAFIA COMPUTARIZADA
ADRIANO RAMOS GONCALVES; KAREN BASSO; PAULO ROBERTO PARTICHELLI; JACOB SCHARCANSKI; MAGALI T. LONGHI
PONTIFICIA UNIVERSIDADE CATOLICA DO RIO GRANDE DO SUL - BRASIL 545

UTILIZACION COMPARADA DE DIVERSOS ALGORITMOS DE VISUALIZACION DE FACETAS EN C.A.D.
O. SANTANA; J.B. ALBERTINI; B. AUBRY; G. MEUNIER; J.L. COULOMB
LABORATOIRE D'ELECTROTECHNIQUE DE GRENOBLE (UA CNRS 355) - FRANCIA..... 546

SISTEMA DE ANIMAÇAO AUXILIADO PELO COMPUTADOR
GEOVANNI DE ARAUJO BERTACO; LUIZ FERNANDO MESQUITADESOUZA; MARCELOROSSONI; J. SCHARCANSKI
PONTIFICIA UNIVERSIDADE CATOLICA DO RIO GRANDE DO SUL - BRASIL..... 548

LOS SISTEMAS DE MANUFACTURACION FLEXIBLE (FMS):UNAIMPLEMENTACION
LIZ JEANNETTE CUBILLOS CANCINO; MARTA LUCIA PARDO MARTINEZ
UNIVERSIDAD NACIONAL DE COLOMBIA - COLOMBIA..... 549

DEFINICION DE LA BASE DE CONOCIMIENTOS PARA UN SISTEMA DE DIAGNOSTICO DE FALLAS EN CONTROL AUTOMATICO DE PROCESOS
DANIEL SBARBARO H.; DANIEL CAMPOS D.
UNIVERSIDAD DE CONCEPCION - CHILE..... 551

APLICACIONES NOVEDOSAS

ASPECTOS DE LA INFORMATICA MEDICA: EVOLUCION DESARROLLO Y PERSPECTIVAS EN COLOMBIA
EMILIO SORZANO ESPINOSA
COLOMBIA..... 552

SISTEMA INTEGRADO DE CONTROL PARA LA UNIDAD DE CUIDADOS INTENSIVOS PEDIATRICA
E. SORZANO E.; C. SANCHEZ FALLA; M. JOSEFA DUQUE;J. ERNESTOCASTRO; F. GONZALEZ FRANCO
COLOMBIA..... 558

SISTEMA AUTOMATICO DE EVALUACION DIAGNOSTICA EN EDUCACION MEDICA
EMILIO SORZANO ESPINOSA; ALBERTO BERNAL; CARLOS RODRIGUEZ
COLOMBIA..... 564

ANALISIS MATEMATICO Y CALCULO DE RIESGO EN ENFERMEDADES GENETICAS
EMILIO SORZANO ESPINOSA; ALBERTO MARTINEZ; MARCELA TRIANA; JAIME BERNAL
COLOMBIA..... 568

MODELO PARA EL CONTROL Y VIGILANCIA EPIDEMIOLOGICA DE LA RABIA
EMILIO SORZANO ESPINOSA; GERMAN PINZON C.; MAURICIO LARA R.
COLOMBIA 572

SISTEMA DE CONTROL DIGITAL PARA EL MUSCULO ESQUELETICO
JORGE ORLANDO CASTRO NOVA; MIGUEL VICENTE PARRADO LOZANO
UNIVERSIDAD NACIONAL DE COLOMBIA - COLOMBIA 577

PROYECTO PILOTO DE INFORMATICA PARA EL MANEJO DE LOS IMPUESTOS EN COLOMBIA
ESPERANZA BEDOYA V.; ALFREDO HERNANDEZ C.
MINHACIENDA - COLOMBIA 597

SISTEMA DE INFORMACION PARA LA ADUANA NACIONAL DE COLOMBIA
ALFREDO HERNANDEZ CIFUENTES; FABIO CAMACHO PARDO
MINHACIENDA - COLOMBIA 610

LA SISTEMATIZACION DE LA SUPERINTENDENCIA BANCARIA COLOMBIANA
ALFREDO HERNANDEZ CIFUENTES; RODRIGO NASSAR MONTOYA
MINHACIENDA - COLOMBIA 629

BASE DE DATOS DE LA DEUDA PUBLICA COLOMBIANA (FENIX)
EDUARDO ASMAR; ALFREDO HERNANDEZ; ALEJANDRO SALAMANCA; ANDRES SUAREZ
MINHACIENDA - COLOMBIA 642

APLICACION DE LOTUS 123 AL MANEJO DE DATOS DE RIEGO
RICARDO SANCHEZ
UNIVERSIDAD DEL VALLE - COLOMBIA 659

SISTEMA UNIVERSITARIO DE INFORMATICA JURIDICA
JESUS SANTOS AMAYA
ICFES - COLOMBIA 668

VOLUMEN II

EDUCACION EN INFORMATICA

AMERICA LATINA: ALTERNATIVA DE LA REVOLUCION INFORMATICA HACIA EL AÑO 2000.
JAIME ACOSTA PUERTA
CRESET - COLOMBIA 670

MODELOS CURRICULARES EN INFORMATICA-COMPUTACION
VICTORIA BAJAR; GUILLERMO LEVINE
INSTITUTO TECNOLOGICO AUTONOMO DE MEXICO - MEXICO 695

INFORMATICA Y COOPERACION: UNA PERSPECTIVA
RICCARDO FERRARI; GERONIMO BELLASAI; BENJAMIN BARAN
UNIVERSIDAD CATOLICA DE NUESTRA SENORA DE LA ASUNCION - PARAGUAY 709

ALFABETIZACION MASIVA EN COMPUTACION EN LA UNIVERSIDAD CATOLICA DE CHILE
PEDRO HEPP K.; LUIS DISSETT
UNIVERSIDAD CATOLICA DE CHILE - CHILE 719

EDUCANDO HACIA LA INGENIERIA DE SOFTWARE: UNA REINTERPRETACION PARA URUGUAY
DANIEL SCHWED
URUGUAY 726

UNA LINEA DE INVESTIGACION EN LOS ESTUDIOS DE INGENIERIA DE SISTEMAS EN LA UNIVERSIDAD NACIONAL DE COLOMBIA.
CARLOS CORTES AMADOR
UNIVERSIDAD NACIONAL DE COLOMBIA - COLOMBIA 738

PLANIFICACION DE INFORMATICA EDUCATIVA A NIVELES UNIVERSITARIOS
CARLOS SCHEEL
UNIVERSIDAD DEL SAGRADO CORAZON - PUERTO RICO 741

INFORMATICA EDUCATIVA

UN AMBIENTE DE DESARROLLO DE SOFTWARE PARA EDUCACION
PEDRO HEPP K.; JAIME NAVON C.
UNIVERSIDAD CATOLICA DE CHILE - CHILE 743

ORIGEN-DESARROLLO E IMPLEMENTACION DE UN SISTEMA COMPUTACIONALDEAPOYOAPROCESOSDE ENSENANZA-APRENDIZAJE
GIOVANNI SENI ZAMBRANO
UNIVERSIDAD DE LOS ANDES - COLOMBIA 754

SISTEMA EDUCATIVO COMPUTACIONAL PARA EL SUMINISTRO DE ANESTESIA
SARA CROITORU EINSTEIN; VIVIAN LUCHAU BOTERO; JORGE MEJIA
UNIVERSIDAD DE LOS ANDES - COLOMBIA 760

CARDIAC: UNA AYUDA DIDACTICA PARA LA ENSENANZA DE QUE ES UNA COMPUTADORA
M. OBREGON S.; S. PATRICIA ALBA S.;J. L. CHAMORRO; MARIA T. OSPINA; G. BALLA; S. VARGAS
COLOMBIA 777

SISTEMA TUTOTIAL CON EJERCITACION Y PRACTICA PARA LA ENSENANZA DE LA SUMA
FERNANDO M. DURAN CAMPOS
UNIVERSIDAD DE LOS ANDES - COLOMBIA 784

SISTEMA DE EDUCACION POR COMPUTADOR SECOM
JOSE ABRAHAM CASTAÑEDA VARGAS; MARIA FERNANDA RAMIREZ DE CASTAÑEDA
COLOMBIA 795

USO DE MINIX PARA LA ENSENANZA DE SISTEMAS OPERATIVOS
MARTIN SANTANA; MIGUEL SANTANA
ESCUELA DE ADMINISTRACION DE NEGOCIOS PARA GRADUADOS. PERU 815

RED DE INTERCAMBIO DE INFORMACION SOBRE INFORMATICA EDUCATIVA
ALVARO H. GALVIS PANQUEVA; OLGA MARIÑO; FRANCISCO RUEDA
UNIVERSIDAD DE LOS ANDES - COLOMBIA 817

SISTEMA MANEJADOR DE BANCOS DE PROBLEMAS
ALVARO H. GALVIS PANQUEVA.; OLGA MARIÑO; FRANCISCO RUEDA F.
UNIVERSIDAD DE LOS ANDES - COLOMBIA 819

SISTEMA DE EVALUACION DEL CONOCIMIENTO DE ESTUDIANTES CON APOYO COMPUTACIONAL
CESAR GUZMAN RENTERIA; CARLOS ALBERTO SANTANA LANCON
INSTITUTO POLITECNICO NACIONAL - MEXICO 821

INVESTIGACION OPERACIONAL

UNA APROXIMACION A LA DINAMICA PERIURBANA
ISAAC DYNER
UNIVERSIDAD NACIONAL DE COLOMBIA - COLOMBIA 822

SISTEMA PARA GENERACION EN MICROCOMPUTADORES DE MATRICES Y REPORTES DE PROGRAMACION LINEAL
B. BRITO; R. CALABRESE; E. CORREA; B. LORENZO
UNIVERSIDAD CENTRAL DE VENEZUELA - VENEZUELA 831

PROGRAMA PARA LA SOLUCION DE MODELOS DE COLAS
MAURICIO ZUÑNIGA CASTAÑEDA; FERNANDO PALACIOS GOMEZ
UNIVERSIDAD DE LOS ANDES - COLOMBIA 850

TIEMPO: SISTEMA PARA ANALISIS GRAFICO DE SERIES DE TIEMPO
F. DA SILVA; E. GRANADOS; B. LORENZO
UNIVERSIDAD CENTRAL DE VENEZUELA - VENEZUELA 871

EVALUACION Y PREDICCION DEL DESEMPENO DE SISTEMAS COMPUTACIONALES
IGNACIO CASAS; FRANCISCO AURTENECHEA
UNIVERSIDAD CATOLICA DE CHILE - CHILE 889

SIMCOL : UNA HERRAMIENTA PARA SIMULAR MODELOS DE REDES DE COLAS
FEDERICO R. ECHAVARRIA R.; FERNANDO PALACIOS GOMEZ
UNIVERSIDAD DE LOS ANDES - COLOMBIA 909

REDES Y SISTEMAS DISTRIBUIDOS

PROTOCOLOS DE CONVERSION PARA EL INTERCAMBIO DE INFORMACION ENTRE BASES DE DATOS BIBLIOGRAFICAS HETEROGENEAS
JOSE ABASOLO PRIETO
UNIVERSIDAD DE LOS ANDES - COLOMBIA 931

TESEO: AMBIENTE DE PROGRAMACION DE SISTEMAS TRANSACCIONALES DISTRIBUIDOS
MARIA CONSUELO FRANKY
UNIVERSIDAD DE LOS ANDES - COLOMBIA 946

COMPILADOR DE EXPRESIONES DEL LENGUAJE RELACIONAL PARA EL SISTEMA TESEO
PERLA LEVY
UNIVERSIDAD DE LOS ANDES COLOMBIA..... 972

DETERMINACION Y RESTAURACION AUTOMATICA DE LOS ESTADOS INTERMEDIOS DE UNA APLICACION
MARCO MARCINELLI
UNIVERSIDAD CATOLICA DE NUESTRA SENORA DE LA ASUNCION - PARAGUAY..... 991

OPTIMIZACION EN TELECOMUNICACIONES
EDUARDO M. LOPEZ SORIANO
ITESM - INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MO - MEXICO..... 1003

ANALISIS DE LA UTILIZACION MAXIMA DE UN CANAL DE COMUNICACIONES EN UN ENLACE VIA RADIO
BRIA OSCAR NORBERTO; DE GIUSTI MARISA RAQUEL
ARGENTINA 1024

ESPECIFICAÇAO DA CAMADA DE SESSAO PARA A REDE-RIO
LUCI PIRMEZ
UNIVERSIDADE FEDERALE DO RIO DE JANEIRO - BRASIL..... 1037

SISTEMA DE EMULACION DE TERMINAL Y TRANSFERENCIA DE ARCHIVOS
HUGO SIN TRIANA; MAURICIO ARANGO GOMEZ
UNIVERSIDAD DE LOS ANDES - COLOMBIA..... 1062

ANALISE E ESPECIFICAÇAO DOS SERVICIOS DE TRANSFERENCIA ACCESO E MANIPULAÇAO DE ARCHIVOS (FTAM)
ESTHER DE CASTRO PACITTI
UNIVERSIDADE FEDERALE DO RIO DE JANEIRO - BRASIL..... 1078

ANALISIS DEL VIDEOTEX COMO SERVICIO TELEMATICO PARA COLOMBIA
HECTOR ALIRIO ROJAS
UNIVERSIDAD NACIONAL DE COLOMBIA - COLOMBIA..... 1092

BASES DE DATOS Y ESTRUCTURAS DE DATOS

COMPORTAMIENTO DEL ARBOL-BD EN LAS FASES CRECIENTE - DECRECIENTE Y ESTACIONARIO
O. SANTANA; O. MAYOR; M. DIAZ; G. LOPEZ
UNIVERSIDAD POLITECNICA DE CANARIAS - ESPAÑA 1093

BUSQUEDA DE LOS M-VECINOS MAS PROXIMOS EN EL ARBOL-BD
O. SANTANA; M. DIAZ; O. MAYOR; J. GONZALEZ
UNIVERSIDAD POLITECNICA DE CANARIA - ESPAÑA 1106

AN ALGORITHM FOR DYNAMIC STORAGE ALLOCATION AND ITS APPLICATION TO B+-TREES
RICARDO A. BAEZA-YATES
UNIVERSITY OF WATERLOO - CANADA 1122

ESTRUCTURAS DE DATOS PLEX
JORGE ARGENGDHI; GUILLERMO CICILEO; SONIA RUEDA; MARCELO ZANCONI
CENTRO REGIONAL DE INVESTIGACIONES BASICAS Y APLICADAS - ARGENTINA 1133

ARBOLES QUINTARIOS: ESTUDIO EXPERIMENTAL PARA INTERROGACIONES EXACTAS- PARCIALES- EN RANGO Y EN RANGO PARCIAL
O. SANTANA; O. MAYOR; M. DIAZ; S. REINA
UNIVERSIDAD POLITECNICA DE CANARIAS - ESPAÑA 1148

ESQUEMAS Y ESTRUCTURAS PARA LA BUSQUEDA DE LAS PALABRAS MAS SIMILARES A UNA DADA
O. SANTANA; M. DIAZ; O. MAYOR; J. REYES
UNIVERSIDAD POLITECNICA DE CANARIAS - ESPAÑA 1169

UM SISTEMA PARA AUXILIO AO PROJETO E INTEGRAÇAO DE ESQUEMAS CONCEPTUAIS SEMANTICOS
JANO MOREIRA DE SOUZA
UNIVERSIDADE FEDERALE DO RIO DE JANEIRO - BRASIL 1190

BASES DE DATOS: OPTIMIZACION DEL MODELO INTERNO
RODOLFO SCHMAL S.; YANKO OSSADON N.
UNIVERSIDAD DE TARAPACA - CHILE 1199

MODELO BORROSO DE UN SISTEMA DE RECUPERACION DE INFORMACION
MIGUEL FRANCISCO BUITRAGO B.; WILLIAM JARAMILLO J.
UNIVERSIDAD EAFIT - COLOMBIA 1208

COMPARACION ESTRUCTURAL DE INGRES Y ORACLE EN EL AMBIENTE UNIX
ROY CAHN-SPEYER V.; ALFREDO HERNANDEZ C.; ANDRES SUAREZ R.
MINHACIENDA - COLOMBIA 1221

SISTEMA ESPECIALISTA PARA VALIDAÇAO DE BASES ARUANDA
ANGELA MARIA BOSISIO; MARCIA PACITTI TELES
SERVICIO GENERAL DE PROCESSAMENTO DE DADOS - SERPRO - BRASIL 1242

SOFTWARE Y HARDWARE DE BASE

EL COMPUTADOR PERSONAL COMO ANALIZADOR DE ESTADOS LOGICOS DE BAJO COSTO
GERONIMO BELLASSAI
UNIVERSIDAD CATOLICA DE NUESTRA SENORA DE LA ASUNCION - PARAGUAY 1252

PROJETO E IMPLEMENTAÇAO DO BANCO DE DADOS DE UN SISTEMA PARA PAC DE MICROCIRCUITOS
KARLA PARREIRAS POLANCZYK; NILTON DE OLIVEIRA JUNIOR; ALBERTO HENRIQUE FRADE LAENDER
UNIVERSIDADE FEDERALE DE MINAS GERAIS - BRASIL 1260

O SOFTWARE DO SISTEMA MULTIPROCESSADOR MICRO-BIS
CELSO MACIEL DA COSTA; SIMAO SIRINEO TOSCANI; THADEU BOTTERI CORSO
UNIVERSIDADE FEDERALE DO RIO GRANDE DO SUL - BRASIL 1273

INFORMATICA TEORICA

INFORMATICA SIMBOLICA

META-CONTROL y PROGRAMACION LOGICA

Adolfo Kvitca
Universidad de Buenos Aires (UBA)
Escuela Superior Latinoamericana de Informática (ESLAI)
Casilla de Correo 3193 - 1000: Buenos Aires - Argentina

Resumen:

En el presente trabajo se describe un formalismo que permite expresar el componente de control de un programa lógico en forma declarativa. Esta representación explícita de la información de control, independiza al sistema del orden parcial de las cláusulas y permite lograr eficiencia en la ejecución, sin sacrificar la inteligibilidad del programa.

La propuesta ilustra principios generales según los cuales diferentes reglas de control puedan ser usadas en forma conjunta, y está inspirada en las ideas sugeridas por R.Davis sobre meta-reglas, pero aplicadas al ámbito de la demostración de teoremas. El sistema permite, a diferencia de otras propuestas, introducir un componente heurístico breadth-first en la búsqueda y expresar diferentes reglas de inferencia de manera natural.

(Nota: La lectura del trabajo presupone conocimientos sobre aspectos relativos al control en programación lógica [KOW/81]).

Este trabajo se ha realizado en el marco del proyecto "Ambiente de Programación Inferencial (API)", financiado por el Programa Nacional de Informática y Electrónica (537-001) SECYT-Argentina

1. Introducción:

En la programación tradicional un programa se realiza especificando las operaciones necesarias para resolver el problema, es decir, diciendo cómo debe ser resuelto, pero quedando usualmente implícitos los supuestos en que el programa está basado. En cambio, en la programación lógica, los supuestos son explícitos, y la secuencia de uso de las operaciones pasa a ser implícita.

La idea clave de la programación lógica es programar mediante descripciones, cambiando de esta manera la perspectiva prescriptiva de la programación hacia una perspectiva declarativa, donde es más importante especificar "qué" debe hacer el programa y no "cómo" debe hacerlo. Por lo tanto, los algoritmos se pueden considerar integrados por un componente lógico, (que especifica "qué" es lo que se debe hacer) y un componente de control (que determina "cómo" debe ser hecho).

La eficiencia de un algoritmo puede ser mejorada modificando el componente de control sin cambiar la lógica y por lo tanto sin cambiar el significado del algoritmo. Un programa puede incluir en el componente lógico lo que otro incluye en el componente de control, pero en general, son preferibles los algoritmos que, para aumentar la eficiencia, ponen el énfasis en el componente de control. Esto permite que el componente lógico sea más claro y que represente, en forma obviamente correcta, el problema y el conocimiento que puede ser usado para su solución. La hipótesis es que, cuando se desarrollan programas bien estructurados mediante refinamientos sucesivos y sistemáticos, el componente lógico debe especificarse antes que el componente de control. Es decir que, primero se debe representar y describir el problema y luego, controlar la búsqueda para que sea resuelto en un lapso razonable.[KOW/79]

2. El Problema del Control en Prolog:

Prolog ha ganado popularidad como lenguaje de programación lógica, [CLO/84],[STE/86] pero sin embargo, no cumple con las altamente deseables características enunciadas. La causa es su rígida estrategia de control para realizar las deducciones ("backtracking"), lo que origina, entre otros inconvenientes, serialidad, ineficiencia y ciclos.

La noción serial impuesta por el backtracking, obliga a tener en cuenta el orden de las cláusulas y de los literales dentro de ellas. Además, el predicado "cut", brindado al programador para que pueda controlar el desarrollo del árbol, es un mecanismo de muy bajo nivel. (Se dice que el "cut" es a la programación lógica como el "go to" a la programación imperativa). Esto no significa que no deben existir mecanismos para decidir qué partes del árbol desechar o probar primero, pero nunca a expensas de la inteligibilidad del programa.

Cuando los programadores se independicen de estas deficiencias, sus intuiciones se moverán en un nivel superior, cambiando la perspectiva empírica (que fomenta "local-patching": cómo hago que esto se haga a continuación), a una perspectiva más general, que considere la estrategia de control como información acerca del nivel objeto.

3. Diferentes Facilidades de Control:

Para solucionar estos inconvenientes se han implementado diversos sistemas con distintas facilidades de control, que corresponden en mayor o menor grado a diferentes propuestas. Las mismas se pueden clasificar en:

.a)Control Incorporado al Programa: implementado mediante anotación de variables (Ej:IC-PROLOG [CLA/80]), metapredicados en cláusulas (Ej: freezer, PROLOG II [COL/82]), distintas conectivas para conjunciones (Ej: EPILOG [POR/84],[PER/84]).

.b)Control Específico: Enfoque seguido principalmente por Bundy [BUN/81], que consiste básicamente en la construcción de intérpretes específicos para el dominio de aplicación.

.c)Control Separado del Programa: utilización de aserciones independientes de las cláusulas del programa para expresar distintos aspectos de control (Ej:MU-PROLOG [NAI/84c], METALOG [DIN/84], CERT [GAL/80]. Naish muestra cómo, para algunos casos, este control puede ser generado automáticamente [NAI/83]).

Estos sistemas proveen algunas soluciones, pero: a) no ofrecen un tratamiento uniforme del problema del control, b) se basan únicamente en un mecanismo depth-first (que no es completo), y c) realizan backtracking "inteligente" utilizando solamente propiedades sintácticas (METALOG provee un mecanismo más general (aunque no lo suficiente), donde se puede especificar a qué resolvente volver en caso de falla).

4. Representación Explícita del Control:

La representación explícita del control en forma separada del componente lógico satisface las caracteristicas deseadas.
La ventaja de especificar el control en forma separada del programa consiste en que el control no está restringido a una cláusula, sino que está relacionado con todo el espacio de búsqueda y permite la expresión de estrategias más generales. El componente lógico del programa es más claro y se puede experimentar con distintos tipos de control sin cambiarlo.

Siguiendo la línea expuesta anteriormente, los programas deben desarrollarse en forma declarativa pero, desde un punto de vista práctico, se puede observar que un estilo puramente declarativo es inadecuado para la demanda de muchas aplicaciones: los procedimientos de entrada/salida y otras acciones que contienen efectos colaterales, se deben realizar eficientemente y en secuencias cuidadosamente diseñadas.

Esta aparente contradicción está originada en el hecho de que hay información que el programador conoce, pero que en el programa está sólo en forma implícita. La hipótesis es que la ineficiencia no depende de una estrategia de deducción particular, sino que es intrínseca al requerimiento de que toda la lógica del programa sea expresada de una manera uniforme y no diferenciada.

5. Propuesta para la Representación Explícita del Control:

La propuesta consiste, entonces, en que el programador especifique explícitamente los siguientes aspectos:

a) Información correspondiente a la selección de la regla:
Las condiciones, relacionadas con la selección de una regla, deben explicitarse para poder evaluarlas (cuando sea posible) estrictamente antes de las consecuencias de la misma:

selector:predicado <- sub-objetivos.

(donde selector y sub-objetivos consisten en una conjunción -posiblemente nula- de predicados). La regla PROLOG equivalente sería: predicado <- selector,sub-objetivos, pero con la diferencia fundamental que, si bien el selector se evalua preferentemente antes de los sub-objetivos, los literales en cada uno de ellos se evaluan mediante una estrategia mas flexible.

b) Las reglas excluyentes:
El conocimiento del programador acerca de la imposibilidad de invocar más de una regla para un predicado particular se expresa de la siguiente manera:

selector:predicado <-> sub-objetivos.

Esta información puede ser usada para controlar que realmente las distintas reglas sean excluyentes, para lograr eficiencia -en caso de satisfacerse una regla no es necesario intentar probar el resto de las alternativas- y finalmente, para expresar claramente la utilización de reglas "default", en el sentido que se utilizan sólo si no hay otra alternativa. La regla PROLOG equivalente sería: predicado:selector,!,sub-objetivos.

Ej: Mediante un selector "else" que tiene la propiedad de tener prioridad mínima para la selección de reglas (es decir que la regla que contiene el selector "else" se utilizará sólo después de haber intentado el resto), se puede implementar naturalmente, una estructura "if-then-else":

```
if-1(x):p(x) <-> then-1(x).
if-n(x):p(x) <-> then-n(x).
else   :p(x) <-> then(x)  .
```

(if-# una serie de condiciones y then-# una serie de acciones).

Ej: La definición de factorial sería entonces:

```
                fact(0 1) <->.
     mayor(X 0):fact(X Y) <->
                +(X1 1 X),fact(X1 Y1),*(Y1 X Y).
```

(En particular, a pesar de la exigencia de evaluar el predicado "mayor" antes del resto, es posible -mediante reglas de control adecuadas-, posponer su evaluación hasta que las variables esten instanciadas).

c) Las reglas que corresponden a acciones:

Consisten en una secuencia ordenada de acciones a tomar cuando se satisface el predicado, o el mismo es requerido en algun sub-objetivo:

```
selector:predicado -> acciones.
```

La utilización de "acciones" también esta gobernada por la regla de exclusión, pero tiene la propiedad adicional de siempre satisfacerse la invocación a la misma, eventualmente mediante una acción nula. (Se asume implícito que en toda acción "p", existe una regla de la forma: else:p(x) ->.). Esto permite implementar acciones "transparentes" ("demonds"), por ejemplo:

```
trace:resolvente(X) -> imprimir("resolvente:",X)
```

d) Los aspectos referentes al control:

La idea consiste en evitar la explosión combinatoria mediante un demostrador de teoremas que piensa acerca de qué está intentando hacer. Para esto los objetivos, acciones y razonamientos deben estar explícitamente representados en una forma manipulable por el proceso de deducción. En rigor, este problema interno de control, puede ser formalizado usando afirmaciones y reglas de la misma manera que se hace en el dominio externo, considerando el problema de deducción desde un punto de vista "meta".

A este aspecto está dedicado el resto del artículo, y consiste en la definición de un formalismo para tratar distintos aspectos referentes al control de una manera explícita y uniforme.

(Nota: En los ejemplos se usará X1.X para denotar la lista con primer elemento X1 y resto X, y X..Y para la lista resultado de concatenar la lista X con la lista Y).

6. Una Visión "Meta" del Problema de Control:

La posibilidad del sistema de manipular aspectos relativos a la dualidad conocimiento / meta-conocimiento, teoría / meta-teoría, lenguaje / meta-lenguaje, permite expresar conocimiento extensivamente y utilizarlo eficientemente [BOW/82], pudiéndose expresar aspectos de gran interés en diversas áreas de la Inteligencia Artificial: en Deducción Automática se implementan estrategias que utilizan meta-teoremas para acortar las pruebas en el nivel objeto (Ej:paramodulación); en Representación del Conocimiento, para aumentar el poder expresivo de los lenguajes de representación, (Ej: puede servir para formalizar creencias, razonamiento "default", no monotonía, etc); en Resolución de Problemas para controlar el espacio de búsqueda. [AIE/86]

A continuación se expondrá un formalismo para expresar el componente de control en forma declarativa, que está inspirado en las ideas de Davis sobre meta-reglas [DAV/80], pero aplicadas al ámbito de la demostración de teoremas. El mismo, pretende introducir principios generales donde diferentes reglas de control puedan ser usadas en forma conjunta. (El análisis se restringe a estrategias lineales (SLD) que son eficientes y, en particular, completas para cláusulas de Horn [LLO/83], pero, es aplicable a estrategias de resolución mas generales).

La idea fundamental del mismo, consiste en identificar en el proceso de deducción, tres fases distintas, y proveer mecanismos para manipular cada una de ellas:

a) recuperación: alguna propiedad de las cláusulas es usada para elegir un subconjunto (RECUPERAR-cláusulas).

b) refinamiento: las ramas, reglas y literales se reordenan y filtran (ORDENAR-ramas,reglas,literales).

c) ejecución: se obtiene un nuevo resolvente (obtener-RESOLVENTE).

El algoritmo corresponde básicamente al siguiente esquema:

```
Demo(teoría,nil.árboles)      <- .
Demo(teoría,árbol1.árboles) <-
        ORDENAR-literales(árbol1,árbol1'),
        RECUPERAR-cláusulas(teoria,árbol1',cláusulas),
        ORDENAR-cláusulas(cláusulas,cláusula1.cláusulas'),
        obtener-RESOLVENTE(árbol1',cláusula1,resolvente),
        armar-arbol(árbol1',cláusulas',árbol1''),
        ORDENAR-ramas(resolvente.árbol1'',árboles,árboles'),
        Demo(teoria,árboles').
```

Observar que siempre se selecciona el primer elemento de cada estructura, pues ésta se encuentra constantemente ordenada y que la utilización de los predicados de las distintas fases no deben instanciar variables de los predicados del nivel objeto.

Los sistemas existentes sólo atacan la fase de refinamiento, y lo hacen en forma parcial pues, están desarrollados, en mayor o menor medida, alrededor de las siguientes ideas:

a) durante la ejecución "forward" debe decidirse qué literal resolver (elección AND) y qué cláusula usar para resolverlo (elección OR).

b) durante el "backtracking", en caso de falla, debe decidirse qué pasos deshacer y cómo seguir.

(En particular, PROLOG, elige el literal más a la izquierda y la primera cláusula que unifique (en orden de escritura) y, en caso de falla, realiza "backtracking" al resolvente precedente en orden cronológico, eligiendo la próxima cláusula que unifique con el mismo literal)

Siendo las principales deficiencias las siguientes:

a) no ofrece un tratamiento uniforme del problema del control (los distintos mecanismos se implementan en formas diferentes, (especialmente el componente "forward" y "backward"), lo que hace difícil, y en muchos casos imposible, la cooperación entre ellos).

b) se basa únicamente en un mecanismo depth-first (que además de no ser completo, utiliza la información heurística en forma parcial, lo que muchas veces determina una complejidad exponencial [PEA/84])

c) la verdadera inteligencia (heurística dependiente del dominio), es puesta en la componente "forward" y realiza "backtracking inteligente" utilizando solamente propiedades sintácticas. [BRU/84],[PIE/82]

d) una vez realizada la elección (AND) de un literal, ésta no tiene posibilidad de ser revisada

6.1. Refinamiento:

La presente propuesta soluciona esas deficiencias y consiste básicamente en considerar al proceso de refinamiento únicamente como depuración y reordenamiento (de un árbol de busqueda particular), basados en aspectos heurísticos provistos por el usuario o generados automáticamente. Este conocimiento heurístico se expresa mediante meta-reglas, que pueden tener un alcance global o local y, que pueden incluir invocación dirigida mediante "content-reference" [DAV/79]. La heurística utilizada puede introducir un componente "breadth-first" en el recorrido del árbol (aunque "depth-first" se asume por "default").

A continuación se ilustrará, mediante una serie de ejemplos, las características de organización del árbol de busqueda, diseñado con el propósito de permitir la revisión de las elecciones "AND" y la incorporación de un componente "breadth-first", características distintivas con respecto a otros formalismos y que permite obtener demostraciones más cortas.

En cada nivel del árbol, se elige un literal y la cláusula correspondiente, dejando asociado al árbol las cláusulas no elegidas que se utilizarán posteriormente.

Ej: <- P(X),Q(Y),R(X)

P(a) <-. Q(c) <-. R(e) <-. S(g) <-.
P(b) <- Q(d) <-. R(f) <-.
P(Z) <- S(a).

Utilizando la "estrategia PROLOG" el árbol quedaría con 9 nodos:

(0) P(X),Q(Y),R(X)!X,Y
(1) Q(Y),R(X)!X/a,Y (2)P(X){P(b),P(Z)<-S(a)},Q(Y),R(X)!X,Y
(11) R(X)!X/a,Y/c (12)Q(Y){Q(d)},R(X)!X/a,Y
falla
(121) R(X)!X/a,Y/d
falla
(21) Q(Y),R(X)!X/b,Y (22)P(X){P(Z)<-S(a)},Q(Y),R(X)!X,Y
(211) R(X)!X/b,Y/c (212)Q(Y){Q(d)},R(X)!X/b,Y
falla
(2121) R(X)!X/b,Y/d
falla
(221) S(a),Q(Y),R(X)!X,Y
falla

Mediante "Backtracking Inteligente" el árbol sería de 6 nodos:

(0) P(X),Q(Y),R(X)!X,Y
(1) Q(Y),R(X)!X/a,Y (2)P(X){P(b),P(Z)<-S(a)},Q(Y),R(X)!X,Y
(11) R(X)!X/a,Y/c (12)Q(Y){Q(d)},R(X)!X/a,Y
falla
(21) Q(Y),R(X)!X/b,Y (22)P(X){P(Z)<-S(a)},Q(Y),R(X)!X,Y
(211) R(X)!X/b,Y/c (212)Q(Y){Q(d)},R(X)!X/b,Y
falla
(221) S(a),Q(Y),R(X)!X,Y
falla

En cambio, mediante la mejor elección permitida en formalismos tradicionales (no permiten revisión de la elección AND), sería de 5 nodos:

(0) P(X),Q(Y),R(X)!X,Y
(1) P(X),Q(Y)!X/e,Y (2) R(X){R(f)},P(X),Q(Y)!X,Y
(11) S(a),Q(Y)!X/e,Y
falla
(21) P(X),Q(Y)!X/f,Y
(211) S(a),Q(Y)!X/f,Y
falla

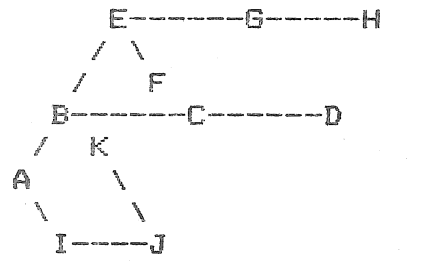
Pero el esquema propuesto permite, eligiendo adecuadamente, reducir el árbol a solo 4 nodos:

(0) P(X),Q(Y),R(X)!X,Y
(1) S(a),Q(Y),R(X)!X,Y (2)P(X){P(a),P(b)},Q(Y),R(X)!X,Y
falla
(21) P(X){P(a),P(b)},Q(Y)!X/e,Y
(22)R(X){R(f)},P(X){P(a),P(b)},Q(Y)
falla
(221) P(X){P(a),P(b)},Q(Y)!X/f,Y
falla

En el ejemplo se puede notar una semejanza del árbol con el modelo sugerido en [NAI/84a] "Heterogenous SLD Resolution (HSLD)", sin embargo el presente sistema fue concebido independientemente. Aunque desde el punto de vista teórico ambas propuestas son equivalentes (únicamente en lo referente a la organización del árbol: en HSLD sólo se contempla la etapa de refinamiento), y por lo tanto, todos los resultados sobre consistencia y completitud se mantienen [LLO/83], la presente formulación conduce a una implementación mucho más eficiente que la permitida por HSLD, pues restringe la invocación de predicados y podría mantener las unificaciones realizadas en el "Máximo Subconjunto Unificable (MUS)" [CHE/86],[COX/84].

A continuación se muestra la conveniencia de la posibilidad de utilización de un componente "breadth-first".

Supongamos el siguiente mapa, representado con las cláusulas correspondientes, donde se desea hallar si es posible ir de A hasta K:




```
ir(a,b)<-.  ir(b,c)<-.  ir(b,e)<-.
ir(a,i)<-.  ir(e,f)<-.  ir(c,d)<-.
ir(e,g)<-.  ir(j,k)<-.  ir(i,j)<-.
ir(g,h)<-.

camino(X,X)<-.
camino(X,Y)<-ir(X,Z),camino(Z,Y).
      <-camino(A,K)
```

Se considera como información heurística la distancia aerea entre las ciudades, es decir, en cada momento se elige la ciudad que este más cerca del objetivo K:

```
dist-k(a,2)<-.  dist-k(b,1)<-.  dist-k(c,3)<-.  dist-k(d,6)<-.
dist-k(e,4)<-.  dist-k(f,2)<-.  dist-k(g,3)<-.  dist-k(h,6)<-.
dist-k(i,2)<-.  dist-k(j,2)<-.  dist-k(k,0)<-.
```

Si esta información se utiliza en forma local, es fácil verificar que las ciudades seran "visitadas" en el siguiente orden: A,B,C,D,E,F,G,H,I,J,K
En cambio, introduciendo una componente "breadth-first" en el uso de la heurística, es decir utilizándola en forma global, se obtiene un comportamiento más eficiente, pues el orden sería: A,B,I,J,K

Un ejemplo más drástico, donde utilizando una estrategia "depth-first" no es posible hallar las dos soluciónes para ningún ordenamiento (inclusive dinámico) de las cláusulas, es el siguiente:

```
<-C(X).
C(X)<-P(X).     C(X)<-Q(X).
P(0).           P(X)<-P(s(X)).
Q(1).           Q(X)<-Q(s(X)).
```

Pues una vez encontrada una solución, el sistema "depth-first" entra en una rama infinita.

A continuación se muestra cómo expresar los meta-predicados sugeridos principalmente por Dincbas [DIN/84] y Gallaire [GAL/80].

```
Selección de literales:
ACTIVATE(R,L) <- condición
                 [activar el literal subsumido por L en el
                 resolvente R]
FREZZE(R,L) <- condición
                 [postergar la utilización de L en R]
BEFORE(L1,L2) <- condición
                 [seleccionar el literal L1 antes del L2]

ORDENAR-literales(nil,nil) <->.
ACTIVATE(X,X1),borrar(X1,X,X'):ORDENAR-literales(X,X1.Y) <->
    ORDENAR-literales(X',Y).
FREEZE(X,X1),borrar(X1,X,X'):ORDENAR-literales(X,Y..(X1)) <->
    ORDENAR-literales(X',Y).
BEFORE(X1,X2):ORDENAR-literales(X1.X,X1.Y) <->
    ORDENAR-literales(X,Y).
BEFORE(X1,X2),borrar(X1,X,X'):ORDENAR-literales(X2.X,X1.X2.Y)<->
    ORDENAR-literales(X',Y).
else:ORDENAR-literales(X1.X,X1.Y) <->
    ORDENAR-literales(X,Y).

Selección de cláusulas:
CHOOSECLAUSE(L,B) <- condición
                 [dar prioridad a la cláusula con cuerpo B para el
                 literal L]
INHIBCLAUSE(L,B) <- condición
                 [evitar la cláusula con cuerpo B para el literal
                 L]
OPBEFORE(C1,C2) <- condición
                 [utilizar la cláusula C1 antes de la C2]

ORDENAR-cláusulas(nil,nil) <->.
CHOOSECLAUSE(L,B),BORRAR(L.B,X,X'):ORDENAR-cláusulas(X,(L.B).Y).
         ORDENAR-cláusulas(X',Y).
INHIBCLAUSE(L,B):ORDENAR-cláusulas((L.B).X,Y) <->
         ORDENAR-cláusulas(X,Y).
OPBEFORE(X1,X2):ORDENAR-cláusulas(X1.X,C1.Y) <->
         ORDENAR-cláusulas(X,Y).
OPBEFORE(X1,X2),borrar(X1,X,X'):ORDENAR-cláusulas(X2.X,X1.X2.Y).
         ORDENAR-cláusulas(X',Y).
else:ORDENAR-cláusulas(X1.X,X1.Y) <->
         ORDENAR-cláusulas(X,Y).
```

(Obviamente, para lograr que "else" sea elegido siempre último sólo hay que escribir: BEFORE(X,else:Y) <-)

```
Selección del punto de "Backtracking":
INHIBACK(L,R) <- condición
                [evitar la elección del resolvente R en caso de
                falla de L]
BACKFAIL(L,R) <- condición
                [utilizar el resolvente R en caso de falla de L]
REJECTGOAL(N,R) <- condición
                [no intentar probar el resolvente R de nivel N]

INHIBACK(L,X1),falla(L):ORDENAR-ramas(nil,X1.X,Z) <->
        ORDENAR-ramas(nil,X,Z).
BACKFAIL(L,X1),falla(L),borrar(X1,X,X'):ORDENAR-rama(nil,X,X1.Z)
        ORDENAR-ramas(nil,X',Y).
else:ORDENAR-ramas(nil,X,X) <->.
REJECTGOAL(N,X1),profundidad(X1,N):ORDENAR-ramas(X1.X,Y,Z) <->
        ORDENAR-ramas(X,Y,Z).

Implementando "Depth-First":
ORDENAR-ramas(X,Y,Z),DIF(X,nil) <->
        FILTRAR-ramas(X..Y,Z).
Implementando "Breadth-First":
ORDENAR-ramas(X,Y,Z),DIF(X,nil) <->
        FILTRAR-ramas(Y..X,Z).
Implementando "Heuristic Search":
ORDENAR-ramas(X,Y,Z),DIF(X,nil) <->
        ordenar-juntar(X,Y,Z'),FILTRAR-ramas(Z',Z).
```

El mecanismo de "backtracking inteligente" sugerido en [BRU/84], que consiste en volver directamente al "literal culpable" cuando se detecta una falla, también se puede realizar mediante este enfoque.

En el mismo, es necesario guardar información durante la ejecución "forward", con la consiguiente sobrecarga, e impidiendo muchas veces, por aspectos de implementación, la interacción con otras reglas de control. En el sistema propuesto, una regla que puede ser vista como "backtracking inteligente (BI)", consiste en, cuando ocurre una falla, depurar todos los resolventes que tengan literales subsumidos por el que falló y aquellos en los que las variables ya hayan tomado su valor (notar que en el árbol están representadas las substituciones en forma explícita y que sería posible agregar información semántica acerca de qué variables son las que habría que tener en consideración para cada predicado). La ejecución "forward" será menos costosa en tiempo y en espacio, no así la "backward", pero al estar el componente "forward" guiado mediante heurística, se necesitaría realizar "backtracking" en menor cantidad de oportunidades. En caso contrario, y si se implementa adecuadamente el árbol de búsqueda, la sobrecarga seguiría siendo lineal.
Ademas, esta regla puede interactuar con otras, en formas que el "BI" no puede, pues este sólo reacciona "inteligentemente" ante los errores cuando estos son descubiertos, en vez de evitarlos en primer lugar [NAI/84a].

"Ordenar una lista" o "el problema de las 8 reinas", son ejemplos de esta situación. Una vez generada una permutación, el "BI" regresa directamente al resolvente culpable de la falla, pero es necesario generar una nueva permutación antes que pueda ser detectada otra falla. Un comportamiento más eficiente estaría dado por la posibilidad de co-rutinar el "test" con la permutación, en forma conjunta con el BI. Es más, en el caso de "ordenar una lista", se podría instrumentar que en el momento de "backtracking" se tenga en cuenta el resolvente donde se instanció la primera variable, pues esta debe ser cambiada de lugar. Actitud que no se podría realizar con el "BI" tradicional.

Es posible también, implementar mecanismos más generales que "aprendan" de los errores, cambiando el orden de los tests, de las cláusulas, etc.

Los métodos de detección de ciclos sugeridos en [BRO/84] también son fácilmente implementables en forma similar al predicado "REJECTGOAL".

6.2. Ejecución:

Expresando explícitamente la fase de ejecución, se puede obtener resultados sumamente interesantes, pues es posible cambiar o ampliar las reglas de inferencia. La regla provista por un sistema típico sería:

a) RESOLVENTE(<- X1.X, X1 <- W, <- W..X) <->.

Pero supongamos que queremos agregar información sobre como actuar con predicados particulares, por ejemplo el predicado "igual": estaremos agregando a la teoría aspectos correspondientes a paramodulación [ROB/69]

b) RESOLVENTE(<- X1=Y1.X, Y2=X1 <- W , <- Y2=Y1.W..X) <->.

Supongamos que se desea demostrar la unicidad del elemento neutro:

<- e=e´ a partir de los siguientes axiomas:
0) X = X <-
1) X.e = X <-
2) e.X = X <-
3) X.e´= X <-
4) e´.X= X <-

utilizando estos axiomas junto con la definición de "RESOLVENTE" de la meta-teoría, quedarían los siguientes pasos:
tomando (3/b): <- e.e´=e´ ; tomando (2/b): <- e´=e´ ; y con (0/a): <-
De esta forma, no solo se obtiene una prueba más corta y más clara, sino que, además se evitan los cíclos debido a la conmutatividad de la suma.

Definir la conmutatividad en la meta-teoría, para cualquier predicado "OP" y de esta manera evitar ciclos, se haría de la siguiente manera:

```
c)  RESOLVENTE(<-X1 OP X2.X,X1 OP X2<-W,<-W..X) <->.
    RESOLVENTE(<-X1 OP X2.X,X2 OP X1<-W,<-W..X) <-> CONMUT(OP).
```

También se puede formalizar el método de resolución para cláusulas generales (con más de una conclusión). Para lograr una estrategia completa, cada vez que se produce un resolvente "merge" éste puede ser agregado a la base de datos.[NIL/80] (Suponer las cláusulas formadas por un conjunto de literales positivos y negativos).

```
d)    RESOLVENTE(  X1.X, Y, Z) <- CANCELAR(X1,Y,Y'),MERGE(X.Y',Z)
<->.
                CANCELAR(+X,-X.Y,Y) <->.
    +X <> Y1:CANCELAR(+X,Y1.Y,Z) <-> CANCELAR(+X,Y,Z).
                CANCELAR(-X,+X.Y,Y) <->.
    -X <> Y1:CANCELAR(-X,Y1.Y,Y1.Z) <-> CANCELAR(-X,Y,Z).
    remueve-repetidos(X,Y):MERGE(X,Y) <-> Agregar(Y).
                          else:MERGE(X,X) <->.
```

Se puede utilizar un parámetro para expresar la probabilidad y computarla mediante meta-cláusulas:

```
e)    RESOLVENTE(  <-  X1.X  / P1, X1 <- Y / P2, <- X..Y.(* P2 P1
P)/P ) <->.
```

Se puede expresar otras reglas de computación, como las sugeridas para "Distributed Logic" [MON/84]:

```
f)  RESOLVENTE(<-(X1)..Y1..(X2)..Y2,X1'X2<-Y,<-Y..Y1..Y2) <->.
```

Los ejemplos muestran como es posible incorporar teorías en el método de resolución, haciendo innecesario resolver directamente a partir de los axiomas de la misma. Esto reduce el largo de las demostraciones y el espacio de prueba, evitando muchas veces ciclos de difícil detección. Esta posibilidad permite considerar al sistema como una implementación del método "Theory Resolution" [STI/85].

6.3. Recuperación:

También se puede explicitar criterios para la fase de recuperación, que están fuertemente ligados con los de la fase de refinamiento:

```
RECUPERAR( X, S) <-> conjunto(clausulas(X <- Y),S).
RECUPERAR( X1=X2, S) <-> conjunto(clausulas(Y1=X1,S)).
```

Inclusive se podría implementar mecanismos de recuperación que utilizaran apareo (matching) aproximado.

También, mediante la implementación de un esquema que mantenga dos sistemas: uno para el objetivo y sus sucesores (sistema "top-down") y otro para los hechos y sus consecuencias (sistema "bottom-up"), es posible lograr estratégias flexibl

es de recuperación. De esta manera, en cambio de existir un único mecanismo predefinido ("goal-directed") el sistema podría cambiar su estructura de control dinámicamente a medida que el programa progresa, utilizando "backward" para algunas demostraciones y "forward" para otras. Por ejemplo:

la recuperación bi-direccional podría expresarse mediante:

use reglas que mencionen el objetivo actual o la conclusión actual

o mediante el criterio sugerido por [POH/72]:

en cada paso elegir la dirección de inferencia que da lugar al menor número de alternativas

la invocación que determina pruebas de "a un paso" mediante:

use reglas que mencionen el objetivo actual y la conclusión actual

Una forma de acelerar algunas deducciones podría consistir en "deducirlas" en el sistema "top-down" y "memorizarlas" en el "bottom-up" para su posterior uso.

7. Limitaciones:

El presente trabajo no abarca la definición de un meta-lenguaje que provea un conjunto de conceptos predefinidos para expresar más facilmente las distintas estrategias. Considero que para el desarrollo del mismo habría que tener en cuenta la clasificación propuesta en [NAI/84b].

Otro aspecto no contemplado concierne al uso de notación funcional y relacional. De otro modo, la siguiente invocación "f(g(x),h(r(y)))" debe escribirse "g(X,U),r(Y,V),h(U,W), f(U,W,Z)", siendo necesario el uso de variables intermedias, e imponiendo nuevos problemas de control. Una posibilidad de solución consiste en "compilar", a partir de la expresión funcional, la forma relacional con el correspondiente control asociado.

Hay que notar que el uso de cláusulas de control no siempre reducirá el tiempo de ejecución, pues hay que agregar el ciclo de interpretación de las cláusulas de control. Sin embargo el crecimiento es lineal y la mejora puede ser exponencial!!. Al respecto, un tema que requiere futura investigación consiste en la posibilidad de utilizar el conocimiento acerca del control para la transformación automática del programa, permitiendo un control más elemental durante la ejecución.

8. Conclusiones:

Se presentó un formalismo que permite expresar el componente de control de un programa lógico en forma declarativa. El mismo tiene un interesante grado de uniformidad, al tratar al problema de control desde el punto de vista "meta", permitiendo la codificación de distintas estrategias específicas que pueden interactuar eficientemente. A diferencia de otras propuestas, ésta posibilita la codificación de estrategias completas, gracias a la introducción de un componente "breadth-first" en la búsqueda.

La hipótesis es que la ineficiencia no depende de una estrategia de deducción particular, sino que es intrínseca al requerimiento de que toda la lógica del programa sea expresada de una manera uniforme y no diferenciada. Esto lleva a explicitar: a)la selección de reglas, b)las reglas excluyentes, c)las acciones secuenciales, y d)la separación de las distintas etapas del proceso de invocación. Estas características brindan al sistema una gran flexibilidad permitiendo expresar fácilmente: a)aspectos de control (refinamiento), b)distintas reglas de inferencia (ejecución), y c)distintas estrategias de búsqueda (recuperación).

Bibliografía:

[AIE/86] L.Aiello,C.Cecchi,D.Sartini,"Representation and Use of Metaknowledge", Proceedings of the IEEE, Vol.74,No 10, 1986

[BOW/82] K.Bowen,R.A.Kowalski,"Amalgamating Language and Metalanguage in Logic Programming", Logic Programming, Clark and Tarnlund(eds), Academic Press, 1982.

[BRO/84] D.R.Brough and A.Walker, "Some Practical Properties of Lógic Programming Interpreters", Proceedings of the 1984 Conference on Fifth Generation Computer Systems, Tokyo.

[BRU/84] M.Bruynooghe, L.M.Pereira, "Deduction Revision by Intelligent Backtracking", Implementations of Prolog, J.A.Campbell (ed), Ellis Horwood, 1984.

[BUN/81] A.Bundy,L.S.Sterling, "Meta-level Inference in Algebra", Workshop on logic programing for intelligent systems, Los Angeles,1981.

[CHE/86] T.Y.Chen,J.L.Lassez,G.S.Port, "Maximal Unifiable Subsets and Minimal Non-Unifiable Subsets", New Generation Computing, 4/86,133-152.

[CLA/80] K.L.Clark and F.G.McCabe, "IC-Prolog - Language Features", in Logic Programming, Clark and Tarnlund(eds), Academic Press, 1982.

[CLO/84] W.F.Clocksin and C.S.Mellish, "Programming in Prolog", Springer Verlag, 1984.

[COL/82] A.Colmerauer,"Prolog-II Manuel de Reference", GIA, Universite d'Aix-Marseille II, 1982.

[COX/84] P.T.Cox, "Finding Backtrack Points for Intelligent Backtracking", Implementations of Prolog, J.A.Campbell (ed), Ellis Horwood, 1984.

[DAV/77] R.Davis, "Generalized Procedure Calling Content Directed Invocation", Proc.Symposium on AI and Programming Languages,Rochester,1977.

[DAV/80] R.Davis, "Meta-Rules: Reasoning about Control", MIT AI Memo 576, 1980.

[DIN/84] M.Dincbas, "Metacontrol of Logic Programs in METALOG", Proceedings of the 1984 Conference on Fifth Generation Computer Systems, Tokyo.

[GAL/80] H.Gallaire and C.Lasserre, "A control Metalanguage for Logic Programming", Logic Programming, Clark and Tarnlund(eds), Academic Press, 1982.

[KOW/79] R.A.Kowalski, "Algorithm=Logic+Control", Communications of the ACM 22, 7/79, 424-436.

[KOW/81] R.A.Kowalski, "Logic for Problem-Solving", North-Holland, 1981.

[LLO/83] J.W.Lloyd, "Foundations of Logic Programming", Springer, 1984.
[MON/84] L.Monteiro, "A Proposal for Distributed Programming in Logic", Implementations of Prolog, J.A.Campbell (ed), Ellis Horwood, 1984.
[PIE/82] T.Pietrzykowski and S.Matwin,"Exponential Improvement of Efficient Backtracking", Proc.6th Conference on Automated Deduction,1982.
[NAI/83] L.Naish, "Automatic Generation of Control for Logic Programs", Negation and Control in Prolog, Springer, 1985.
[NAI/84a] L.Naish, "Heterogeneous SLD Resolution", Negation and Control in Prolog, Springer Verlag, 1985.
[NAI/84b] L.Naish, "PROLOG Control Rules", Negation and Control in Prolog, Springer Verlag, 1985.
[NAI/84c] L.Naish, "MU-PROLOG 3.2db Reference Manual", Negation and Control in Prolog, Springer Verlag, 1985.
[NIL/80] N.Nilsson, "Principles of Artificial Intelligence", Tioga, 1980.
[PEA/84] J.Pearl, "Heuristics, Intelligent Search Strategies for Computer Problem Solving", Addison Wesley, 1984.
[PER/84] L.M.Pereira, "Logic Control With Logic", Implementations of Prolog, J.A.Campbell (ed), Ellis Horwood, 1984.
[POH/72] I.Pohl, "Bi-directional search", Machine Intelligence 7, Edinburgh Uviversity Press, New York.
[POR/84] A.Porto, "EPILOG:A Language for Extended Programming in Logic", Implementations of Prolog, J.A.Campbell (ed), Ellis Horwood, 1984.
[ROB/69] G.Robinson,L.Wos,:Paramodulation and Theorem-proving in First Order Theories with equality, Machine Intelligence 4, 103-33,1969.
[STE/86] Sterling L,E.Shapiro,"The Art of Prolog",MIT Press,1986.
[STI/85] M.E.Stickel,"Automated Deduction by Theory Resolution", Journal of Automated Reasoning 1, 1985.

BACKTRACKING INTELIGENTE PARA PROLOG

SILVIA TAKAHASHI

UNIVERSIDAD DE LOS ANDES
Apartado Aereo 4976

RESUMEN

En la ejecución normal de un programa Prolog, cuando falla un objetivo se reactiva el objetivo anterior. Esto puede ocasionar una pérdida de tiempo si el objetivo anterior no ayuda a solucionar la falla. En este caso, el interpretador terminará por saltar más atrás al terminar de explorar todas las posibilidades del objetivo al cual se devolvió. Si solamente se devuelve a objetivos que tengan posibilidad de resolver la falla, se debe producir un ahorro considerable de tiempo.

Se podría pensar entonces, que un interpretador de Prolog que tuviera esta facilidad sería muy superior a uno sin ella, pero pesando los problemas que trae la implementación con las ventajas que produce, esto no es muy claro. El problema radica en que el programador puede escribir sus programas de manera que no se vea la diferencia entre uno y otro. Todo depende de la eficiencia de los programas.

0. Introducción.

Este artículo explica el backtracking inteligente aplicado a la interpretación de PROLOG [ROU73]. Además analiza la dificultad de la implementación de esta forma de hacer bactrack sobre una máquina intermedia Prolog con estructura de pilas. Finalmente analiza las ventajas y desventajas de agregar esta mejora a los interpretadores de Prolog.

M. Bruynooghe [BRU78], [BRU80] propuso el backtracking inteligente como una forma de hacer bactrack en programas lógicos cortando las recontestaciones inútiles.

Este trabajo está basado en la tesis de grado de autora [TAK87] en la que se describe la implementación del bactracking inteligente sobre la máquina intermedia Prolog descrita por D. Warren [WAR77] e implementada por R. Arias en su tesis [ARI85].

El lector debe estar familiarizado con conceptos básicos de Prolog tales como unificación, bactrack y corte. Además debe conocer las principales características de la máquina descrita por Warren.

1. Definición de Términos.

Un programa Prolog se compone de una base de datos y de un objetivo inicial. La base de datos es una sucesión de cláusulas de Horn de dos tipos, hechos y reglas. Las reglas se componen de una cabeza y un cuerpo separados por el símbolo ":-" y terminan con un punto: cabeza:-cuerpo. . Los hechos son reglas que no tienen cuerpo: cuerpo. . El objetivo inicial puede verse como una cláusula que no tiene cabeza; es un cuerpo precedido por el símbolo "?-" : ?-cuerpo. .

Las cabezas son fórmulas atómicas y los cuerpos son sucesiones de fórmulas atómicas separadas por comas, que en lo sucesivo llamaremos subobjetivos o llamadas.

Como es bien conocido, el lenguaje Prolog tiene dos semánticas [BRU78]: la declarativa y la procedimental. Utilizándolas se puede analizar la corrección de los programas escritos en Prolog. La semántica declarativa se basa en el significado lógico de las cláusulas y la procedimental en el control de ejecución.

La semántica que se necesita para poder describir un algoritmo de interpretación es la procedimental. Utilizando esta semántica las cláusulas que componen la base de datos son definiciones de procedimientos y los subobjetivos que conforman el cuerpo de las cláusulas y el objetivo inicial son llamadas a estos procedimientos. El objetivo inicial es el programa principal.

El conjunto de cláusulas cuyas cabezas tienen el mismo nombre constituye la definición del procedimiento con ese nombre. A las cabezas de estas cláusulas se les da el nombre de puntos de entrada del procedimiento. El hecho de que haya más de un punto de entrada para algunos procedimientos da la posibilidad de hacer backtrack para encontrar la solución de un objetivo inicial.

Los subobjetivos se van resolviendo en orden secuencial de izquiera a derecha y las cláusulas para resolverlos se van escogiendo en orden descendente.

A grandes rasgos, la ejecución de un programa Prolog se hace en la forma descrita a continuación.

Supóngase el siguiente objetivo inicial:

?-a1, ... , ar. (r $\geq$ 0).

En este momento este es el objetivo actual.

Se toma la primera llamada del objetivo actual a1 y se busca el primer punto de entrada para esa llamada, la cláusula b :- c1, ... ,cm. (m $\leq$ 0), que cumpla con la condición de que los argumentos de b y de a1 unifiquen con un unificador más general θ. Esto da como resultado otro objetivo:

?- (c1, ... ,cm,a2, ..., ap.) θ

A este objetivo lo llamamos objetivo derivado y al proceso aplicado lo denominamos proceso de avance.

Ahora el objetivo derivado es el actual.

El proceso de avance se aplica repetidamente sobre los objetivos derivados. Este ciclo se detiene cuando se llega a un objetivo derivado vacío o cuando no se encuentra una cláusula que unifique con el primer subobjetivo del objetivo derivado.

Si se da el primer caso, se dice que el programa termina exitosamente. Se contesta sí y como resultado se dan los valores que deben tomar las variables del objetivo inicial para que éste se cumpla. Estos valores se obtienen haciendo la composición de todas las substituciones aplicadas.

Si se da el segundo caso, se debe hacer backtrack . Para explicar este concepto necesitamos hacer algunas convenciones. Al objetivo que, al aplicarle el proceso de avance, dio como objetivo derivado el objetivo actual, lo llamamos punto de backtrack y a la cláusula aplicada en este proceso de avance la llamamos cláusula errónea . El proceso de hacer backtrack consiste, entonces, en devolverse al punto de backtrack y tratar de volver a aplicar el proceso de avance.

Vemos que la ejecución de un programa Prolog describe una lista de objetivos que se obtienen aplicando repetidamente el proceso de avance. Hacer back - track se reduce a devolverse al objetivo anterior.

Cuando decimos resolver el objetivo i, recontestar el objetivo i o falló el objetivo i nos estamos refiriendo a la primera llamada del i-ésimo objetivo.

Si se da el caso de que el objetivo inmediatamente anterior no se pueda recontestar porque la cláusula que se utilizó para resolverlo era el último punto de entrada de la definición de ese procedimiento, decimos que este objetivo es un objetivo determinístico [PER82]. Una definición más formal de este concepto es la siguiente: "Un objetivo es determinístico si su primera llamada después de resuelta no tiene posibilidad de ser recontestada." Para ahorrar tiempo, al fallar un objetivo debe devolverse, en lugar de al objetivo inmediatamente anterior, al objetivo no determinístico más reciente. Este es ahora su punto de back track.

Otra definición que también es necesaria es la del concepto de padre de un objetivo. El i-ésimo objetivo es padre del j-ésimo objetivo si al pasar del objetivo i al i+1 se generó la primera llamada del objetivo j. Es decir, esta llamada formaba parte del cuerpo de la cláusula que se utilizó para resolver el objetivo i.

Cuando hay un corte, el problema se reduce a marcar como determinísticos los objetivos desde el actual hasta su padre. Es decir, si el corte es la primera llamada en el objetivo i y el padre de éste es j entonces todos los objetivos desde i hasta j son marcados como determinísticos. Si falla algún objetivo después del corte el interpretador se comporta como si fallase la cláusula que lo activó.

2. Arbol de ejecución.

Para facilitar la explicación de algunos conceptos, en este documento se describe la ejecución de un programa Prolog por medio de un árbol que se va creando durante la ejecución.

Un nodo es un subobjetivo y sus hijos son los subobjetivos de la cláusula utilizada para resolverlo. Si esta cláusula es un hecho, el hijo del subobjetivo es un nodo que llamamos hoja vacía . La raiz de este árbol es un subobjetivo ficticio que tiene como hijos a los subobjetivos del objeto inicial.

Vemos entonces, que tenemos tres tipos de nodos: la raiz, los subobjetivos o llamadas y las hojas vacías.

A cada nodo de tipo llamada le corresponde lo siguiente:
- El conjunto de las variables que aparecen en la llamada.
- El conjunto de las cláusulas que conforman la definición del procedimiento y que no han sido desechadas. En un principio incluye a todas las cláusulas en la definición.
- La cláusula que fue aplicada para resolver el subobjetivo. Si todavía no se ha resuelto no está definida.
- El conjunto de las cláusulas que han sido desechadas ya sea porque no unificaron o porque se hizo backtrack al objetivo en el cual ese subobjetivo era la primera llamada y se escogieron para resolverlo.

A estos conjuntos los llamamos VARS , DISP , APLIC y ELIM respectivamente.

Vemos que en cualquier momento de la ejecución y para cualquier llamada a se cumple que:

$$ELIM \cap \{APLIC\} \cap DISP = \emptyset$$
$$ELIM \cup \{APLIC\} \cup DISP = \text{Conjunto de las cláusulas que componen la definición del procedimiento para } a .$$

Los nodos terminales de tipo llamada representan las llamadas que todavía no se han resuelto. En un momento dado de la ejecución, el objetivo que se está tratando de resolver está formado por estos nodos terminales. A partir de esto podemos tener la lista de objetivos que se van derivando durante ejecución.

Si el objetivo actual es i debemos conocer además de i a todos los objetivos menores a i. Además para cada objetivo debemos conocer su padre, su punto de backtrack y, si ya se resolvió, las substituciones hechas para pasar al siguiente objetivo.

3. Motivación.

La idea fundamental de hacer backtracking inteligentemente en Prolog consiste en evitar que al fallar un objetivo i en el interpretador se devuelva a un objetivo j cuya recontestación no ayude a cambiar el hecho de que i falle. Además al devolverse no deben deshacerse llamadas cuyas ejecuciones no se vean afectadas por la recontestación de ese objetivo.

Se ve la necesidad de usar backtracking inteligente en el siguiente ejemplo.

Dado el siguiente programa Prolog:

```
1. g1(X1) :- g(X1,X1).
2. g(X2,Y2) :- f(X2,Z2),h(Y2,W2).
```

```
3. g(a,Y3) :- l(Y3).
4. l(a).
5. f(a,b).
6. f(a,c).
7. h(n,m).

?-g1(a)
```

La lista de objetivos que se van derivando es:
1. g1(a). unifica con la cláusula 1.
2. g(a,a). unifica con la cláusula 2.
3. f(a,Z2),h(a,W2). unifica con la cláusula 5.
4. h(a,W2). falla,el último objetivo no determinístico es el 3.
3. f(a,Z2),h(a,W2). unifica con la cláusula 6.
4. h(a,W2). falla, el último objetivo no determinístico es el 2.
2. g(a,a). unifica con la cláusula 3.
3. l(a). unifica con la cláusula 4.
4. EXITO

Vemos que el devolverse al objetivo 3 al fallar el objetivo 4, no cambió el hecho de que éste fallara. Si se hiciese backtracking inteligentemente se ahorraría este paso inútil porque se daría cuenta de que el objetivo 3 no tiene posibilidad de solucionar la falla y se devolvería directamente al objetivo 2.

La recontestación de algunos objetivos no sirve. Si se devuelve a un objetivo que no ayuda a solucionar la falla, en algún momento se acaban las posibilidades para recontestarlo y termina devolviéndose más arriba. Haciendo backtracking inteligentemente se ahorran estos pasos inútiles.

Si al tratar de resolver un objetivo i, no tenemos éxito, debemos devolvernos solamente a objetivos de los cuales dependa i. Es decir, a objetivos que tengan alguna posibilidad de remediar la causa de la falla.

4. Dependencia. [BRU78] , [BRU80]

La primera llamada de un objetivo i existe y algunas de las variables en sus argumentos tienen ciertos valores debido a la aplicación del proceso de avance a ciertos objetivos para resolverlos. Diremos que el objetivo i depende de estos objetivos y al conjunto de éstos lo llamamos D-Set. Denotamos el conjunto de dependencias del i-ésimo objetivo D-Set(i).

Si un objetivo falla sólo debemos hacer backtrack a objetivos dentro de su D-Set. Escogemos uno de ellos, digamos el más reciente, y lo reactivamos.

La relación de dependencia es una relación transitiva. Sólo es necesario guardar las dependencias directas en el D-Set. Vemos entonces que i depende de j si j pertenece al D-Set(i) o existe k en D-Set(i) tal que k depende de j .

La definición del conjunto de dependencias debe darse operacionalmente. Este conjunto se va creando durante ejecución y es más natural este tipo de definición.

Supóngase que estamos tratando de resolver el objetivo i. Inicialmente su conjunto dependencias incluye únicamente a su padre. Obviamente un objetivo depende de su padre. El conjunto se va aumentando durante la ejecución del algo-

ritmo de unificación. Supóngase que estamos unificando uno de los argumentos de la primera llamada de i con un argumento de la cabeza de la cláusula con la cual se está tratando de resolver. Si es necesario acceder a substituciones hechas al resolver un objetivo j distinto de i y del cual i no depende, para encontrar el valor de uno de sus argumentos, debemos agregar j al conjunto de dependencias de i. Cada vez que se agregue un elemento deben eliminarse los elementos que son redundantes por transitividad. Otra simplificación de los conjuntos de dependencia se obtiene no guardando en ellos objetivos determinísticos. Si en un momento debemos agregar a D-Set(i) un elemento j y j es un objetivo determinístico no lo agregamos a D-Set(i); en lugar de hacer esto unimos D-Set(i) con D-Set(j) y a esta unión le quitamos las redundancias. Esto se justifica para no perder soluciones.

Otra clase de dependencia debe ser tenida en cuenta. Supóngase que estamos en un objetivo i y que éste falla. Hacemos backtrack a un objetivo j que esté en D-Set(i) y lo reactivamos. Estamos tratando de solucionar este objetivo debido a la falla de i, entonces D-Set(j) debe modificarse. Esta modificación es la siguiente:

$$D\text{-}Set(j) := D\text{-}Set(j) \cup D\text{-}Set(i) - \{j\}$$

Después las redundancias deben ser eliminadas.

5. Más definiciones.

Basándonos en el árbol de ejecución descrito en el parágrafo 2, daremos otras convenciones y definiciones necesarias para describir el algoritmo de interpretación con backtracking inteligente. Llamamos Conjunto de Entrada [BRU80] de un objetivo i y de una cláusula (digamos la j-ésima en la base de datos), con la cual se resolvió o trató de resolver el objetivo, al conjunto de objetivos que pueden ser responsables de una eventual falla. Es como un conjunto de dependencias para la unificación. Denotaremos este conjunto de entrada como Input(i,j). Utilizando el árbol de ejecución como referencia, vemos que debe haber un conjunto de entrada para cada cláusula APLIC y para las cláusulas en ELIM de cada subobjetivo. Para cada objetivo, su conjunto de dependencias es la unión de los conjuntos de entrada de la cláusula APLIC y de las cláusulas en ELIM de su primer subobjetivo. Además su D-Set debe incluir a su padre. Estos conjuntos pueden simplificarse para no guardar dependencias indirectas ni objetivos determinísticos.

Una definición más exacta del conjunto de entrada de un objetivo i y de una cláusula j se obtiene definiendo para las variables otro conjunto, que llamaremos LIG [PER82]. Este conjunto se define así: pertenecen al conjunto LIG los objetivos que dieron valor a esa variable al resolverlos. Por ejemplo, si una variable X está ligada al valor a(I,O) e I está ligada a q y O a s, LIG de X debe contener los objetivos que al resolverse ocasionaron que X, I y O se ligaran a los valores que tienen ahora. Además de esto, si al resolver un objetivo j se ligaron dos variables que pertenecían a un mismo subobjetivo, j debe estar en LIG de ambas variables. El último caso se da cuando en la cabeza de una cláusula aparece una variable más de una vez y unifica con dos variables libres diferentes.

El conjunto LIG se va modificando durante la ejecución del algoritmo de unificación. Si durante esta ejecución se le da valor a una variable, se debe agregar a LIG el objetivo que se está tratando de resolver. Si se liga una variable con valor a una libre entonces se debe copiar el conjunto LIG de la ligada en el de la libre. Si se ligan dos variables que están en la misma llamada y ambas están libres, se debe agregar al LIG de ambas el objetivo que se está tratando de

resolver.

El conjunto de entrada de un objetivo con una cláusula también se va creando durante la ejecución del algoritmo de unificación. Cuando se esté unificando una variable con un valor, se une el conjunto LIG de esa variable con el conjunto de entrada y cuando se unifican dos variables y ninguna está libre se agrega el conjunto LIG de cada una al conjunto de entrada.

6. Un algoritmo de unificación. [TAK87]

Lo más importante para un algoritmo de interpretación de Prolog que hace backtracking inteligentemente es el algoritmo de unificación.

Supongamos que:
1) El objetivo actual es el i-ésimo.
2) Estamos tratando de unificar su primera llamada, o1 con la cabeza de la k-ésima cláusula en la base de datos.
3) Tenemos el conjunto de dependencias de i, D-Set(i).
4) Los argumentos de la llamada son X1, ..., Xn.
5) Los argumentos de la cabeza de la cláusula con la que se está tratando de unificar son Y1, ... , Yn.

Después de la ejecución del algoritmo de unificación debemos analizar dos casos. Si tuvo éxito la unificación se deben hacer efectivas las substituciones generadas por la unificación. Si falló la unificación, las substituciones que se hayan alcanzado a hacer deben ser desechas. En ambos casos, se debe crear el conjunto de entrada para el objetivo i y la cláusula k y aumentar el conjunto de dependencias de i con este conjunto de entrada.

Este algoritmo de unificación se diferencia de uno normal, únicamente en que en él se deben tener en cuenta las modificaciones de los conjuntos LIG e INPUT descritas en la sección anterior.

7. La interpretación.

Habiendo dado las convenciones y descrito el algoritmo de unificación sólo resta explicar el ciclo de ejecución para el interpretador.

No hay cambio en el proceso de avance; solamente se presentan problemas cuando hay que devolverse al fallar.

El interpretador descrito en el parágrafo 1 escoge como punto de backtrack al último objetivo no determinístico. Este interpretador debe escoger, para devolverse, un objetivo en su conjunto de dependencias.

En [BRU78] se prueba que cualquier objetivo en el conjunto de dependencias puede ser escogido como punto de backtrack sin causar pérdida de soluciones. Esta observación es válida para programas lógicos en los cuales no se especifica el orden en que se solucionan las llamadas ni el orden en que se escogen las cláusulas. En el caso de Prolog, se debe devolver al objetivo más reciente, ya que de otra forma la semántica procedimental se vería afectada. Es claro que el devolverse al objetivo más reciente no cambia en nada la ejecución. Supóngase que falle el i-ésimo objetivo y que el (i-3)-ésimo es el objetivo más reciente en D-Set(i). No recontestar los objetivos i-1 e i-2 no cambia el ciclo de ejecución, ya que el recontestarlos no arregla la falla. Cuando haya pasado por todas

las recontestaciones de i-1 , trata de recontestar i-2 y para cada una de sus recontestaciones vuelve a explorar las recontestaciones de i-1. Como ninguna de éstas arregla la falla, finalmente se devuelve al objetivo i-3. Devolverse directamente ahorra tiempo sin alterar el resultado.

En [ERU80] se afirma que al fallar un objetivo i y devolverse a tro objetivo j , no se deben deshacer las ejecuciones de todos los objetivos desde i hasta j .

Esta modificación hace que sea necesario tener una lista de objetivos explícitamente. Digamos que al tratar de resolver el i-ésimo objetivo fallamos y que escogimos el j-ésimo como punto de backtrack. Como no se deshacen todas las ejecuciones, el objetivo que se reactiva no es el mismo j-ésimo. Para simular esto no nos devolvemos; creamos un objetivo siguiente (i+1) el cual es una copia del j-ésimo quitando las llamadas cuya ejecución debe mantenerse. Cada una de estas llamadas fue resuelta en un objetivo k ($j < k < i$). Si este objetivo no depende de j su ejecución debe mantenerse. Aún si no se mantiene su ejecución podemos no sacar todas las llamadas del conjunto ELIM, de su primer subobjetivo, para pasarlas al conjunto DISP del mismo. De pronto la recontestación no cambia el hecho de que estas cláusulas no unifiquen. Para saber si debemos sacarlas de ELIM y pasarlas a DISP debemos conocer sus conjuntos de entrada. Debemos sacar de ELIM aquellas cláusulas en cuyos conjuntos de entrada esté j (directamente o por transitividad). Para mantener la semántica procedimental, si sacamos la k-ésima cláusula, debemos sacar también todas las cláusulas que estén definidas antes de k en la base de datos del conjunto de entrada. Esta parte del backtracking inteligente consiste en cambiar el orden de ejecución cuando no se ve afectada la solución final para aumentar así la eficiencia.

8. La implementación. [TAK87]

La última parte explicada del backtracking inteligente no fue implementada, pues habría sido necesario cambiar totalmente la filosofía de la máquina intermedia Prolog (MIP). En la implementación no es necesario hacer copias de los objetivos al hacer backtrack, simplemente se devuelve. Tampoco necesitamos estructuras de datos para los conjuntos de entrada. No son necesarios, pues la parte no implementada del backtracking inteligente es la que, basándose en este conjunto, decide cómo modificar los conjuntos ELIM de los subobjetivos del punto de backtrack. Este es el único momento en que se usan. De resto siempre se utiliza el conjunto de dependencias que se obtiene de la unión de todos los conjuntos de entrada con el padre del objetivo. Entonces , en el algoritmo de unificación modificamos directamente el conjunto de dependencias.

La dificultad principal radicó en el diseño de las estructuras de datos. Es claro que es mucho más natural una estructura de árbol que una de pilas para la implementación del backtracking inteligente. Fue necesario definir una nueva estructura como una lista de objetivos,para que el algoritmo fuera más sencillo y eficiente. Sin embargo así se aumentó considerablemente la cantidad de memoria utilizada. La implementación de los conjuntos LIG influyó también en el aumento de la cantidad de espacios necesarios para cada variable en la pila.

9. Ventajas y Desventajas. TAK87

La implementación del backtracking inteligente sobre un interpretador cualquiera de Prolog presupone un aumento en la cantidad de memoria utilizada.

Veamos que debe tener un interpretador de estos en contraposición con uno

normal. Se deben tener los conjuntos de dependencias para cada objetivo, los conjuntos de entrada para cada llamada en ELIM y el conjunto LIG para cada variable. El costo en espacio es grande; habría que ver que tanto se gana en tiempo. Utilizando este método se ahorran muchos pasos inútiles. Si la memoria no es un problema, sí vale la pena utilizarlo. En caso de tener limitaciones de memoria, no se ve muy claro si se justifica su uso.

En muchos de los casos en los que utilizando el backtracking inteligente, se presenta un comportamiento distinto al obtenido sin este método, esto se puede evitar cambiando el orden de los subobjetivos en las definiciones de las cláusulas. Al fin y al cabo el backtracking inteligente simula el hecho de que los subobjetivos estén en distinto orden. Los cortes también pueden ser utilizados si se desea que un objetivo no se devuelva al anterior. Otra forma de programar eficientemente se logra definiendo procedimientos auxiliares.

Si se programa cuidadosamente es posible que nunca se vean las ventajas del backtracking inteligente. En este caso, se habría pagado un costo alto en espacio sin recibir retribuciones en tiempo. Para decidir si vale la pena usarlo es necesario ver que tan frecuentemente se ve una diferencia en el comportamiento de los dos métodos. Todo depende del programador al cual está dirigido. Una ventaja obvia es que el programador no tiene que preocuparse porque el orden le afecte la eficiencia de sus programas ya que el interpretador hace este trabajo por él.

10. Conclusiones.

El backtracking inteligente usado para la interpretación de Prolog fue modificado para no entrar en conflicto con la semántica procedimental.

La implementación de esta mejora sobre una máquina intermedia con estructura de pilas fue un proceso complicado debido a que la descripción del algoritmo dada en [BRU78] y [BRU80] se amolda más a una estructura de árbol.

Por último es bueno decir que el backtracking inteligente, sólo se justifica en un número reducido de programas, ya que si se programa eficientemente, el comportamiento de un interpretador con backtracking inteligente y de uno sin él no se diferencian. Si la memoria es un recurso barato, valdría la pena implementarlo para liberar al programador de la preocupación por la eficiencia.

11. Referencias.

[ARI85] Arias R., Compilación de cláusulas Prolog. Tesis de Grado. U. de los Andes. 1985.

[BRU78] Bruynooghe M., "Intelligent Backtracking for an Interpreter of Horn Clause Logic Programs", Mathematical Logic in Computer Science, Proceedings of the Colloquium Salgotarjan, Hungary, Sept. 1978.

[BRU80] _____, "Analysis of Dependencies to Improve the Behavior of Horn Clause Logic Programs", Afdeling Toegepaste Wiskunde en Programmatie Katholieke Universiteit Leuven. 1980.

[PER82] Pereira, L.M. 7 Porto, A., "Selective Backtracking", Logic Programming, Academic Press, 1982.

[ROU73] Roussell, PROLOG, Manuel de Reference et d'utilisation. Groupe Intelligence Artificielle, Université Marseille, 1975.

[TAK87] Takahashi, S., Backtracking Inteligente para un Interpretador de Prolog. Tesis de Grado. Universidad de los Andes. 1987.

[WAR77] Warren D. et al., "Prolog: The Language and its Implementation compared with LISP", Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages, ACM 1977.

PROGRAMACION LOGICA BASADA EN ASERCIONES
Una herejía imperativa

por

JAIME A. BOHORQUEZ V.

Departamento de Sistemas y Computación.
Universidad de Los Andes
Aptdo Aéreo 4976
Bogotá
COLOMBIA

Resumen: Se propone una interpretación de la ejecución de un programa lógico en términos de un esquema de transición de estados. Con base en esta interpretación se define la noción de corrección total basada en especificaciones expresadas mediante aserciones, y se desarrolla un cálculo de corrección en el estilo de la lógica de Hoare [H69]. Se dan ejemplos ilustrativos.

Palabras claves: Programación lógica, aserciones, lógica de Hoare, corrección total, lenguaje PROLOG.

0. Introducción.

La programación lógica consiste en la interpretación de la lógica de predicados como un paradigma de programación (alternativo a la programación imperativa y a la programación funcional) para el cual cierto tipo de fórmulas (llamadas cláusulas) constituyen un lenguaje de programación.

El primer lenguaje lógico llamado PROLOG [C73], [R75] fué diseñado e implantado en 1972 y se basó en la interpretación procedimental de las cláusulas de Horn debida a Kowalski [K74]. Posteriormente se diseñó un compilador de PROLOG (escrito en PROLOG) que fué implantado por Warren, Pereira y Pereira [W77], mostrando que el compilador PROLOG ejecutaba programas lógicos tan eficientemente como LISP compilado.

En la programación lógica se expresa únicamente la lógica de los métodos de solución de los problemas, en contraste con los programas convencionales, los cuales mezclan la lógica del conocimiento usado en solucionar los problemas, con el control de los métodos que procesan la información. Por esta razón se afirma que los programas lógicos son fáciles de entender, verificar y cambiar y que serían especialmente adecuados para programadores principiantes y usuarios de bases de datos que no quieran saber de los detalles sobre el control necesario para ejecutar sus programas.

Sin embargo, hay un precio que pagar por el logro de las ventajas mencionadas anteriormente, y es a costa de la eficiencia del programa obtenido al tratar la lógica simplemente como un lenguaje de especificación sin tener en cuenta sus aspectos pragmáticos (ver [K79] cap. 5).

Para el programador profesional la eficiencia es muchas veces un factor de importancia para juzgar la bondad de un programa, lo que muchas veces implica una pérdida de claridad en la interpretación de las cláusulas que lo constituyen. Esto sin detrimento de la prioridad e importancia que tiene la corrección en el desarrollo de programas.

En consecuencia, a pesar de las cualidades autodescriptivas propias de la programación lógica, en ocasiones se hacen necesarias técnicas suplementarias de documentación y verificación de ciertos programas lógicos cuya claridad ha sido sacrificada en aras de un desempeño eficiente. Tal situación se presenta especialmente cierto cuando los programas lógicos se implantan en una máquina secuencial.

Adicionalmente, muchas veces durante la labor misma del diseño de un programa lógico eficiente, el programador familiarizado con la programación imperativa encuentra que una solución sencilla y eficiente para el problema que tiene entre manos es, por ejemplo, la de un ciclo o proceso iterativo; debido al espíritu declarativo característico de la programación lógica, se ve entonces abocado a un bloqueo mental causado por el hecho de estar pensan-

do una solución propia del estilo imperativo de programar.

El propósito de este artículo es el de ayudar al programador de marras a salir de su problema mediante la interpretación de una clase propia de programas lógicos por medio de una semántica operacional basada en una máquina de transición de estados. Esta semántica permitirá el desarrollo de soluciones que se podrán interpretar como iterativas. Además, los programas pertenecientes a la clase mencionada serán suceptibles de documentación por medio de aserciones, y también de verificación mediante un cálculo axiomático inspirado en reglas de inferencia en el estilo de Hoare [H69].

La organización de este artículo será la siguiente: primeramente se presentará la sintaxis y nociones básicas de la lógica clausal. En la sección 2 se explican las nociones de sustitución, unificación, asociación y estado. La sección 3 da una interpretación procedimental de un conjunto de cláusulas de Horn como programa lógico. En la sección 4 se presenta el concepto de corrección de un procedimiento lógico basada en una noción de estado que corresponde a la colección de valores asociados a las variables del procedimiento. La sección 5 describe una semántica operacional que permite interpretar un subconjunto propio de programas lógicos como procedimientos iterativos. En la sección 6 se exhibe un calculo inferencial para demostrar la corrección de un procedimiento lógico. Finalmente en la sección 7 se presentan aplicaciones y ejemplos ilustrativos.

1. La Lógica Clausal.

Un identificador es una cadena finita de símbolos o caracteres alfanuméricos, no todos ellos numéricos.

Una variable se representa mediante un identificador que comience con una letra minúscula:

x a x4 an25

Una constante es cualquier número entero o identificador que no comienza con letra minúscula:

123 Abc 1a MB Persona Sumando

denotarán objetos resultantes de la conceptualización de un problema.

Un término es una variable, una constante o una expresión $f(t_1,...,t_m)$ $(m \geq 1)$ donde f es un identificador (que comienza con minúscula) llamado símbolo de función y los t_i son a su vez términos:

x 123 edad(Jorge) inverso(123)

Una fórmula atómica (o simplemente un átomo) es una expresión de

la forma $p(t_1,...,t_m)$ donde p es un identificador que corresponde a un símbolo de relación n-aria y $t_1,..,t_m$ es una secuencia posiblemente nula (n=0) de términos, usada para expresar relaciones entre términos:

invierta(x) concatene(cons(u,v),L,z)

Una <u>expresión</u> es un átomo o un término.

Una <u>expresión explicita</u> es cualquier expresión que no involucra variables.

Un <u>hecho</u> es un átomo, v.gr. $p(t_1,t_2,...,t_m)$. En este caso se dice que $p(t_1,...,t_m)$ es un hecho acerca de la relación p. Un hecho se interpreta como una afirmación o hipótesis, con todas sus variables cuantificadas universalmente, que puede ser usada para deducir otro hecho o contestar una pregunta.

Una <u>regla</u> consta de dos partes: un átomo llamado <u>cabeza</u> y una lista ordenada de átomos llamada <u>cola</u>. Notacionalmente, en la regla $B:- A_1,..,A_n$. B es la cabeza y $A_1,..,A_n$ es la cola. Si el símbolo asociado a la cabeza de la regla es p, se dice que es una regla acerca de (la relación) p.
Una regla con variables $x_1,..,x_k$ tiene la siguiente interpretación: para todos los valores de $x_1,...,x_k$ vale que lo afirmado por la cabeza de la regla es cierta si se puede establecer la verdad de cada una de las condiciones en la cola de la regla.
Procedimentalmente, una regla $B:- A_1,...,A_n$ se interpreta como un comando condicional o guardado para ejecutar la serie de llamadas de procedimiento $A_1,...,A_n$ cuya condición o guarda es B.

Ejemplo: Si la cabeza de una regla es el átomo abuelo(x,y) y su cola corresponde a la secuencia progenitor(x,z), progenitor(z,y) esta es una regla acerca de la relación "ser abuelo de" y se denota de la siguiente manera:

```
abuelo(x,y):- progenitor(x,z),progenitor(z,x).
```

Una <u>cláusula de Horn</u> es una regla o un hecho.

Una <u>regla recursiva</u> es una regla cuya cola incluye por lo menos un átomo con el mismo identificador que el de la cabeza de la regla, v.gr.

```
p(x) :- q(x,A), p(A), r(x1), p(x).
```

Una <u>regla recursiva terminal</u> es una regla recursiva para la cual únicamente el identificador de el último átomo de su cola coincide con el identificador de su cabeza, v.gr.

```
concat(cons(u,v),y,cons(u,z)):- concat(v,y,z).

p(x):- q(x,x1), p(x).
```

Una <u>lista de objetivos</u> es una sucesión finita de atomos

$<Q_1,\ldots,Q_s>$. Esta noción corresponde a una pregunta o consulta cuya respuesta debe deducirse mediante inferencias, de los hechos y reglas con que se cuenta. Cada uno de los átomos que conforma la consulta se llama un objetivo. La consulta correspondiente a la lista de objetivos $<Q_1,\ldots,Q_s>$ se denota ?- $Q_1,\ldots,Q_s$.

Un procedimiento lógico se denota mediante un símbolo de relación n-aria ($n\geq 0$) (su identificador) seguido de una sucesión de n símbolos de variable llamados parámetros digamos $p(x_1,\ldots,x_n)$, y está compuesto de una secuencia finita de cláusulas acerca de p, llamada el cuerpo de p. Un procedimiento lógico se interpreta como la conjunción de sus cláusulas. Procedimentalmente, el cuerpo de p corresponde a la declaración del procedimiento.

Un programa lógico es un conjunto de procedimientos lógicos. Se interpreta como la conjunción de las cláusulas pertenecientes a los procedimientos lógicos que las conforman.

2. Sustituciones, asociaciones y noción de estado.

Una sustitución S es un conjunto finito de la forma $\{v_1/t_1,\ldots,v_n/t_n\}$ donde cada v_i es una variable y cada t_i es un término diferente de v_i y además $v_i \neq v_j$ para $i\neq j$.

Una sustitución corresponde a una función del conjunto de todas las variables en el conjunto de todos los términos donde la imagen de una variable cualquiera v es t_i, si $v=v_i$ y $v_i/t_i \in S$, en otro caso es v misma.
La sustitución correspondiente a la función identidad se denota por {}.
Si S es una sustitución de la forma $\{v_1/t_1,\ldots,v_n/t_n\}$ y e una expresión, entonces Se es una expresión que se obtiene de e mediante el remplazo simultáneo de cada ocurrencia de la variable v_i ($1\leq i\leq n$) en e por el término t_i. Se dice entonces que Se es una instancia de e.

Ejemplo Si S= {x/A, y/f(B), z/C} y e = p(x,y,z) entonces

$$Se = p(A,f(B),C)$$

Si $e_1,e_2,\ldots,e_m$ son expresiones y S es una sustitución se dice que S es un unificador de $e_1,\ldots,e_m$ si $Se_1 = Se_2 = \ldots = Se_m$. Se dice entonces que $e_1,\ldots,e_n$ son unificables.

Por ejemplo e_1= p(A,y) y e_2= p(x,f(B)) son unificables mediante la sustitucion unificadora {x/A, y/f(B)}.

Un unificador U de las expresiones $e_1,\ldots,e_m$ es un unificador más general si para cualquier unificador S de $e_1,\ldots,e_m$ existe una sustitución T tal que $S = U\circ T$.
Por ejemplo si e_1= p(z,f(z),f(u)) y e_2= p(A,x,f(g(y))) la sustitución {z/A, x/f(A), u/g(y)} es un unificador más general para e_1 y e_2.

Una asociación es una pareja ordenada con una variable como primer elemento, y un término como segundo elemento.

Un ambiente o estado es un conjunto de asociaciones cuyos primeros elementos no se repiten.

Si (v,t) pertenece a un ambiente E se dice que v está ligada a t en E.

El valor de una expresión e en un ambiente A, denotado $val_A(e)$, es el resultado de remplazar cada variable en e que esté ligada en A a un término t por el valor de t ($val_A(t)$) en A. Observe que si una variable no está ligada a ningún término en un ambiente A, su valor en A es ella misma.

Ejemplo: el valor de la expresión p(x,f(y,g(z,x))) en el estado

{(y,f(v,w)), (z,g(x,u)), (u,h(x))} es

p(x,f(f(v,w),g(g(x,h(x)),x)))

3. Interpretación procedimental (secuencial no-determinística).

Una interpretación procedimental secuencial no-determinística de un programa lógico expresado mediante cláusulas de Horn procesa en orden sus objetivos, pero selecciona sus cláusulas de una manera no-determinística.

Un programa lógico se activa mediante la consulta de una lista de objetivos iniciales digamos Q1,...,Qn y continua usando sus procedimientos para derivar listas de objetivos nuevas de viejas. Tal derivación de listas de objetivos se realiza mediante un proceso llamado resolución (que se interpreta como una llamada de procedimiento):

Se obtiene no-determinísticamente del programa una cláusula de la forma Q:- P1,...,Pm ($m \geq 0$) tal que Q unifique con Q1 (en el sentido de que algun unificador mas general S hace a Q y Q1 idénticos) y se deriva una nueva lista de objetivos de la forma

<S(P1),S(P2),...,S(Pm),S(Q2),...,S(Qn)>.

Este proceso se continua aplicando recursivamente con este nuevo objetivo hasta obtener un objetivo vacío.
Si durante la ejecución del proceso que se acaba de describir fuese imposible encontrar una regla que unifique con el primer átomo de cualquiera de los objetivos generados se dice que la ejecución falla y el programa termina con una respuesta negativa.

Observe que en este planteamiento no se considera la posibilidad de reintentar (backtrack) otras posibilidades después que la ejecución haya fallado.

El proceso secuencial de resoluciones de los objetivos con selección no-determinística de cláusulas unificables anteriormente descrito, corresponde en la programación tradicional a la noción de computación propuesta por Dijkstra[D76] con su lenguaje de comandos guardados, donde las condiciones (guardas) de un comando alternativo son seleccionadas no-determinísticamente. La decisión de optar por este tipo de interpretación se toma por la sencillez que se logra para el análisis de correción total de estos programas. Como en el caso de la programación imperativa, estos programas se implantan facilmente en términos de programas con selección secuencial de reglas.

La noción de computación asociada corresponde a la sucesión de listas de objetivos $L_1, L_2, \ldots, L_n$ obtenida a partir de un programa lógico P y una lista inicial de objetivos L_1, donde cada L_{i+1} se obtiene de L_i mediante la resolución del primer objetivo de L_i usando una regla de P (seleccionada no-determinísticamente).

El resultado de la computación es el valor final de las variables en la lista de objetivos original L_1 las cuales resultan afectadas por las sustituciones unificadoras más generales asociadas a las resoluciones realizadas.

Una lista de objetivos se trata como una pila de llamadas de procedimientos. El átomo seleccionado es aquél en el tope de la pila. Las nuevas llamadas de procedimiento que por el proceso de resolución remplazan la llamada de procedimiento seleccionada, se insertan en el tope de la pila para así obtener una nueva lista de objetivos.

Más formalmente, si se define una <u>situación</u> s como un par (O,A), donde O es una lista de objetivos y A un ambiente, entonces

Dados: un programa lógico P,

una situación <u>inicial</u> $s = (O,A)$, con $O = \langle Q \rangle$ tal que Q es un objetivo con variables en $\bar{v}$

El par (P,s) se denomina una <u>consulta</u> (simple).

Una <u>computación</u> C[P,Q,A] de la consulta (P,s) es una sucesión posiblemente infinita de situaciones

$$C[P,Q,A] = \langle s_0, s_1, \ldots \rangle$$

definida así:

a) $s_0 = s$

b) Para $k \geq 0$, si $s_k = (O_k, A_k)$, con $O_k = \langle Q_1, \ldots, Q_r \rangle$

Si r=0 : s_{k+1} no se define, y se dice que C[P,s] <u>termina</u> (exitosamente).
En este caso, se define el <u>resultado</u> de la computación como

$$C[P,O,A](\bar{v}) = val_{Ak}(\bar{v})$$

Si $r>0$: es posible escoger no-determinísticamente en P una regla $O:- R_1,\ldots,R_m$ tal que O y O_1 unifiquen mediante un unificador más general σ, se define

$s_{k+1} = (O_{k+1}, A_{k+1})$ así:

$O_{k+1} = \langle \sigma R_1,\ldots,\sigma R_m, \sigma O_2,\ldots,\sigma O_r \rangle$

$A_{k+1} = \sigma \circ A_k$

En otro caso no se define s_{k+1} y se dice que C[P,s] falla.

Ejemplo 3.1 El siguiente es un programa lógico para concatenar dos listas. El término cons(x,y) se interpreta como una lista cuyo primer elemento es x y cuya cola (resto de lista) es y (abreviado [x|y]). La constante Nil denota la lista vacía (o también []).

```
concat(Nil,w,w).
concat(cons(x,y),z,cons(x,u)):- concat(y,z,u).
```

concat(x,y,z) denota la relación: "z es la lista resultante de concatenar la lista x con la lista y".

Más claramente, usando las abreviaciones se tiene:

```
(1) concat([],w,w).
(2) concat([x|y],z,[x|u]):- concat(y,z,u).
```

Para calcular el resultado de concatenar la lista [2] a la lista [1] el programa se activa mediante la situación inicial s = (O,A) con O = <concat([1],[2],v)>

A = {}

$\bar{v}$ = v

Una computación asociada C podría ser ($s_k = (O_k, A_k)$)

O_0 = <concat([1],[2],v)>

A_0 = {}

Ahora : concat([1],[2],v) unifica con la cabeza de (2) mediante el unificador más general
σ_1 = {x/1, y/[], z/[2], v/[1|u]}

Entonces:

O_1 = <concat([],[2],u)>

$A_1 = \sigma_1 \circ \{\}$

Después: concat([],[2],u) unifica con la cabeza de (1) mediante el unificador más general σ_2 = {w/[2], u/[2]}

Así:

$O_2 = \square$ (donde $\square$ denota una secuencia nula de objetivos)

$A_2 = \sigma_2 \circ \sigma_1 \circ \{\}$

C termina y s_3 no se define .

El resultado de la computación es:

$$C(v) = val_{A2}(v) = [1|[2]]$$

4. Noción de corrección de un procedimiento lógico.

La corrección de un procedimiento lógico se expresará en términos de predicados de algún metalenguaje usado como lenguaje de especificación de entradas (datos) y salidas (resultados) y en general del estado de las variables del procedimiento durante su ejecución.

Si α es un predicado del metalenguaje de especificación que incluye unicamente variables en $\bar{v}= v_1,\ldots,v_n$ (denotado $\alpha(\bar{v})$) entonces

$$(\alpha \mid v_1,\ldots,v_n : t_1,\ldots,t_n)$$

denota el predicado que se obtiene de la sustitución simultanea de todas las ocurrencias de v_i en α por el término t_i. Si $\bar{t}$ denota la secuencia de términos $t_1,\ldots,t_n$ entonces se puede abreviar a $(\alpha \mid \bar{v}:\bar{t})$

Se entenderá por aserción un predicado sobre (los valores de) las variables de un ambiente dado.

Si P es un programa lógico y Q es un objetivo con variables $\bar{v}$ se dice que la consulta ?-Q es totalmente correcta con respecto al programa P, a una aserción de entrada (o precondición) y a una aserción de salida (o postcondición) $\beta(\bar{v})$ notado

$$\{\alpha\}\ (?- Q)/P\ \{\beta(\bar{v})\}$$

si y solo si $(\beta \mid \bar{v}:\ C[P,Q,A](\bar{v}))$ vale en todo ambiente A para el cual vale α, es decir β satisface el resultado de la computación C[P,Q,A] para todo ambiente A en el cual vale α.

Si un programa lógico P contiene un procedimiento lógico con identificador r cuya lista de parámetros $\bar{v}$ se puede dividir en párametros de datos $\bar{x}$ y de resultados $\bar{y}$ se dice que r es totalmente correcto con respecto a P, a una precondición $\alpha(\bar{x})$ y una postcondición $\beta(\bar{x},\bar{y})$ denotado

$$\{\alpha\}\ r(\bar{x},\bar{y})/P\ \{\beta\}$$

si y solo si para toda instanciación explícita $\bar{X}$ de $\bar{x}$ y todo vector de variables $\bar{z}$ se cumple

$$\{(\alpha|\bar{x}:\bar{X})\}\ (?\text{-}r(\bar{X},\bar{z}))/P\ \{(\beta|\bar{x},\bar{y}:\bar{X},\bar{z})\}$$

es decir toda consulta de un objetivo $r(\bar{X},\bar{z})$ es correcta con respecto a $(\alpha|\bar{x}:\bar{X})$ y $(\beta|\bar{x},\bar{y}:\bar{X},\bar{z})$ para toda secuencia $\bar{X}$ de términos explícitos y toda secuencia $\bar{z}$ de variables que correspondan respectivamente a $\bar{x}$ y $\bar{y}$.

5. Semántica operacional en términos de transición de estados.

De ahora en adelante se estudiará una clase especial de procedimientos lógicos: aquellos cuyas reglas recursivas son todas terminales.
Un procedimiento lógico se llama iterativo si sus reglas recursivas son terminales.
Es más nos interesan procedimientos iterativos (digamos) $p(\bar{x},\bar{y})$ para los cuales los parámetros en $\bar{x}$ son usados para transmitir datos y los de $\bar{y}$ para transmitir resultados. Consecuentemente se supondrá que toda consulta al procedimiento p es de la forma

$$?- p(\bar{X},\bar{z})$$

donde $\bar{X}$ es un vector de términos explícitos y $\bar{z}$ vector de variables. No se pierde mucha generalidad si por simplicidad se supone además que el último átomo de toda regla recursiva acerca de p es de la forma $p(\bar{t},\bar{w})$ con $\bar{w}$ conformado unicamente por variables.
Los procedimientos iterativos anteriormente descritos serán denominados especiales.
Con el propósito de dar una interpretación imperativa a un procedimiento lógico especial, se presentará en seguida una semántica operacional que interpretará tanto la consulta inicial del procedimiento como las fórmulas atómicas que conforman cada una de las cláusulas de su cuerpo, en forma de comandos o instrucciones de tipo imperativo. Los efectos de estos comandos se reflejarán en transformaciones del ambiente de ejecución, mediante asociaciones de términos a las variables y parámetros de dicho procedimiento, realizadas por comandos de asignación definidos a continuación.

Primeramente se definirán comandos básicos para alterar un estado o ambiente de ejecución.
Si v es una variable y t un término en el cual v no ocurre, la acción de ejecutar el comando

$$v := t$$

(asignación del término t a la variable v) en el estado E es la de añadir la asociación (v,t) al conjunto de asociaciones que conforman el estado E previa eliminación de cualquier asociación en E cuyo primer elemento sea v.

Se define también un comando de asignación especial

$$v :=$$

cuyo efecto es el de dejar "libre" (su valor es ella misma) a la variable v, su acción sobre el ambiente en que se ejecuta, es la de eliminar de éste cualquier asociación cuyo primer elemento sea v.

Similarmente, si $\bar{v}$ y $\bar{t}$ son respectivamente vectores de variables diferentes $v_1,\ldots,v_n$ y términos $t_1,\ldots t_n$ para los cuales v_i no aparece en t_i entonces

$$\bar{v} := \bar{t}$$

se interpreta como un comando de asignación simultánea, cuyo efecto al ser ejecutado en un estado E es el de añadir las asociaciones (v_i, t_i) al conjunto de asociaciones de E previa eliminación de cualquier asociación en E cuyo primer elemento sea alguno de los v_i.

La ejecución de un procedimiento P con identificador $p(\bar{x},\bar{y})$ perteneciente a un programa lógico PL, se puede entonces visualizar como un proceso iterativo con variables globales $\bar{x}$, $\bar{y}$, $\bar{y}_1$ que corresponden respectivamente a los parámetros de datos, de resultados, y de resultados locales (obtenidos después de "procesar" una cláusula seleccionada para resolver un objetivo) y que se inicia con la invocación de un objetivo de la forma

$$?- p(\bar{X},\bar{z})$$

(donde $\bar{X}$ es un vector de términos explícitos y $\bar{z}$ vector de variables donde se obtendrán los resultados de ejecución) cuyo efecto inmediato es la realización de la asignación simultánea

$$\bar{x},\bar{y} := \bar{X},\bar{y}_1$$

A continuación se selecciona no-deterministicamente una cláusula del cuerpo de P cuya cabeza unifique con el atomo $p(\bar{X},\bar{y}_1)$. Dicha unificación ocasiona instanciaciones de las variables en la cláusula escogida, así como de los parámetros en $\bar{y}_1$, (resultados locales que se van acumulando en los parámetros $\bar{y}$, los cuales guardarán los resultados definitivos) que alteran el estado de ejecución y pueden explicarse en términos de comandos de asignación de variables.

El anterior proceso de resolución lleva a una sucesión (posiblemente nula) de llamados a ejecución de procedimientos de PL "externos" a P (las debidas a los átomos presentes en la parte derecha de la cláusula seleccionada) la cual puede culminar con una nueva invocación (recursiva) de P, $p(\bar{H},\bar{w})$ que conduce a que $\bar{x}$ tome el valor de $\bar{H}$ y $\bar{y}_1$ juegue el papel de $\bar{w}$, después de lo cual se selecciona de nuevo una cláusula de P para ser resuelta con $p(\bar{H},\bar{w})$ continuando el proceso conforme al ciclo ya descrito.

En caso de que la sucesión de llamadas a ejecución de procedimientos externos a P no fuera sucedida por una invocación recursiva de P, y de que dichas llamadas fueran realizadas exitosamente tendríamos una terminación normal del procedimiento P cuyos resultados podrían apreciarse después de la asignación $\bar{z} := val(\bar{y})$, en los valores de las variables $z_1,\ldots,z_n$ correspondientes al ambiente final de ejecución.

La ejecución de P falla si durante alguna iteración del ciclo que se acaba de describir, no es posible realizar la unificación correspondiente, o cuando la llamada de alguno de los procedimientos externos falla. Una terminación anormal se inter-

preta como una respuesta negativa a la pregunta acerca de la existencia de variables $\bar{z}$ que cumplan una cierta especificación preestablecida para el objetivo ?- $p(\bar{X},\bar{z})$.

Dentro de este contexto la ejecución de un procedimiento lógico $p(\bar{x},\bar{y})$ (invocado usando a $\bar{x}$ como parámetros de datos y a $\bar{y}$ como parámetros de resultados) se puede describir por medio de las siguientes 3 operaciones básicas:

-Invocación (inicial o recursiva) de un objetivo correspondiente a un átomo con identificador p.

-Resolución de un átomo correspondiente a $p(\bar{x},\bar{y})$ con una cláusula del cuerpo de $p(\bar{x},\bar{y})$.

-Llamada de procedimiento lógico "externo" (i.e. distinto de $p(\bar{x},\bar{y})$).

Estas operaciones interactuan en un ciclo de máquina como el siguiente. Supongamos que el objetivo a satisfacer es $p(\bar{X},\bar{z})$ donde como antes, $\bar{X}$ es una sucesión de términos explícitos que corresponden a los datos de la consulta, y $\bar{z}$ (sucesión de variables), donde se obtienen sus resultados.

Paso 0. (Invocación inicial) Efectuar las asignaciones

$\bar{x},\bar{y} := \bar{X},\bar{y}_1$ /* en $\bar{y}$ se calculan los resultados finales del procedimiento */

Paso 1. (Resolución)

Si existe una cláusula $C \bullet p(\bar{u},\bar{v}) :- Q_1,\ldots,Q_m$ donde $\bar{u}$ es una secuencia de términos con variables $r_1,\ldots,r_e$ que se resuelve con $p(val(\bar{x}),\bar{y}_1)$
entonces para reflejar el efecto de la unificación de $p(\bar{u},\bar{v})$ y $p(val(\bar{x}),\bar{y}_1)$, efectúe las asignaciones

$\bar{r},\bar{y}_1 := \bar{s},\bar{v}$;
/* donde $\bar{s} = s_1,\ldots,s_e$ son los subtérminos de $val(\bar{x})$ correspondientes a los de $\bar{r}$ de acuerdo a la unificación realizada. */
si no falle.
/*el procedimiento falla, dando una respuesta negativa*/

Paso 2. (Llamada de procedimientos externos)

Ejecute en orden (comenzando por Q_1) los procedimientos Q_i cuyos símbolos de relación sean diferentes a p (la igualdad solo se puede dar con Q_m, el último átomo de C). Estas llamadas pueden afectar las variables en $\bar{v}$ y otras locales pertenecientes a la cláusula C. Si alguna de estas ejecuciones falla (termina con una respuesta negativa) falle.

Si todas estas ejecuciones terminan normalmente entonces
Si C es una regla recursiva (digamos $Q_m \bullet p(\bar{t},\bar{w})$) con $\bar{t}$ secuencia de términos y $\bar{w}$ una secuencia de variables (que normalmente ocurren en $\bar{u}$ y $\bar{v}$ respectivamente) entonces vaya a Paso 3.
sino vaya a Paso 4.
sino falle

Paso 3. (Invocación Recursiva)

Efectue
$\bar{x} := val(\bar{t})$; /* reinicia a $\bar{x}$ */
$\bar{y} := val(\bar{y})$ /* independiza $\bar{y}$ de variables intermedias*/
$\bar{y}_1 := _$; /* limpia $\bar{y}_1$ */
$\bar{w} := \bar{y}_1$; $\bar{y} := val(\bar{y})$; /* sustituye a $\bar{w}$ por $\bar{y}_1$ en $\bar{y}$. (útil si alguna variable de $\bar{w}$ ocurre en $\bar{v}$)*/

Elimine del ambiente todas las asociaciones cuyo primer elemento es una variable local a la regla $p(\bar{u},\bar{v}) :- Q_1,\ldots,Q_m$.
Vaya al Paso 1.

Paso 4. (Terminación) /* se recibe en $\bar{z}$ los resultados de la ejecución */

$\bar{z} := val(\bar{y})$;
Limpie el ambiente (excepto por $\bar{z}$);
Termine.

Ejemplo 5.1 Dado el procedimiento lógico concat, especificado como en el ejemplo 4.1:

{$x_1 = X_1$, $x_2 = X_2$ y X_1, X_2 son listas} concat(x_1, x_2, y) {$y = X_1 * X_2$}

cuyo cuerpo está conformado por las siguientes cláusulas

```
(a) concat([],u,u).
(b) concat([p|t],u,[p|w]):- concat(t,u,w).
```

y el siguiente objetivo:

```
?- concat([A],X2,z).
```

donde X_2 representa una lista cuya descripción exacta no tiene relevancia en este contexto, se explicará la obtención en z de la lista resultante de la concatenación de [A] y X_2 por medio del ciclo de máquina descrito anteriormente.

Claramente $\bar{x}$ corresponde en este caso a las variables x_1 y x_2, así como $\bar{X}$ a los términos explícitos [A] y X_2, y $\bar{z}$ a la variable z entonces

en el paso 0
$\bar{x}, \bar{y} := \bar{X}, \bar{y}_1$ corresponde a la instrucción $x_1, x_2, y := [A], X_2, y_1$ llegandose al estado

(x_1 , [A]) (x_2 , X_2) (y , y_1);

en el paso 1 se resuelve el objetivo concat([A],X_2,y_1) con la cláusula (b) dando lugar a la unificación de $\bar{u}$ con val($\bar{x}$) y de val($\bar{y}$) con $\bar{v}$ es decir, de [p|t],u con [A],X_2 y de y_1 con [p|w]; ésto queda reflejado en la asignación

$p, t, u, y_1 := A, [], X_2, [p|w]$ que corresponde a $\bar{r}, \bar{y} := \bar{s}, \bar{v}$ en el ciclo de máquina; el estado ahora es

(x_1 , [A]) (x_2 , X_2) (p , A) (u , X_2) (t , [])
(y_1 , [p|w]) (y^2, y_1^2)

ahora $\bar{y} :=$ val($\bar{y}$) corresponde a y:= [A|w] el ambiente se actualiza conforme a este cambio:

(x_1 , [A]) (x_2 , X_2) (p , A) (u , X_2) (t , [])
(y_1 , [p|w]) (y^2 , [A|w]) ;

el paso 2 corresponde aquí a una acción nula puesto que concat no invoca procedimientos externos, el ambiente por tanto permanece igual; las instrucciones ejecutadas en el paso 3 son

```
y1:=  ;
x1,x2:= [],X2;
w:= y1;
y:= [A|y1];
```

más la eliminación del ambiente de las asociaciones de variables diferentes de x_1, x_2, y. El estado resultante es entonces

$$(x_1 , []) \quad (x_2 , X_2) \quad (y , [A|y_1])$$

volviendo ahora al Paso 1 la cláusula (a) (concat([],u,u)) se resuelve con concat([],X_2,y_1) con las siguientes asignaciones:

$\bar{r},\bar{y}_1 := \bar{s},\bar{v}$	que corresponde a	$u,y_1 := X_2,u$
$\bar{y} := val(\bar{y})$	que corresponde a	$y := [A|X_2]$

el estado ahora es

$$(x_1 , []) \quad (x_2 , X_2) \quad (u , X_2) \quad (y_1 , u)$$
$$(y , [A|X_2])$$

siendo la cláusula (a) un hecho, la ejecución termina en el Paso 4 que deja en z el valor $[A|X_2]$; el ambiente final es entonces,

$$(z, [A|X_2])$$

6. Cálculo inferencial de la corrección de un procedimiento.

El numeral anterior ilustra la posibilidad de dar una explicación en términos de transición de estados de la ejecución de un procedimiento de un programa lógico. Esto brinda la oportunidad no solo de definir la corrección de un procedimiento sino también de anotar su cuerpo mediante aserciones lógicas.

Por ejemplo una regla recursiva terminal de un procedimiento de un programa lógico podría especificarse de la siguiente manera:

$$\{\alpha\}\ p(\bar{u},\bar{v}) :- Q_1,\ldots,Q_{m-1},p(\bar{t},\bar{w})\ \{\beta\}$$

donde α y β son predicados lógicos de especificación, $\bar{u}$, $\bar{v}$, $\bar{t}$ son secuencias de términos y $\bar{w}$ es una secuencia de variables de la misma longitud que $\bar{v}$; $\bar{u}$ y $\bar{t}$ son de la misma longitud, y normalmente $\bar{t}$ ocurre en $\bar{u}$.

La interpretación procedimental de esta especificación sería: si el ambiente de ejecución del procedimiento lógico P cumple con el predicado α exactamente antes de resolver dicha regla con un objetivo cuyo primer átomo unifica con $p(\bar{u},\bar{v})$, entonces el procedimiento p ejecuta normalmente hasta el momento en que los objetivos creados por $Q_1,\ldots,Q_{m-1}$ ya han sido satisfechos y se debe satisfacer el objetivo correspondiente a la llamada recursiva $p(\bar{t},\bar{w})$, llegandose después de lo cual, a un estado que cumple con lo estipulado por el predicado β.

Más aún, es posible anotar cada una de las cláusulas de Horn de un procedimiento (digamos $q :- r_1,\ldots,r_s$) de la siguiente forma

$$\{\alpha\}\ q :- \{\gamma_1\}\ r_1\ \{\gamma_2\} \ldots \{\gamma_s\}\ r_s\ \{\beta\}$$

Se podría como se hizo arriba, dar definiciones procedimentales de estas anotaciones en términos de la semántica operacional de las operaciones básicas (invocación, resolución y llamada de procedimiento lógico externo) con que se explicó imperativamente la ejecución de un procedimiento.

Sinembargo, es preferible explicar la noción de corrección en términos de una semántica axiomática que se apoye en un conjunto de axiomas (en realidad esquemas de axiomas) y reglas de inferencia en el estilo del cálculo desarrollado por Hoare [H69].

Las reglas de inferencia se describirán como

$$\frac{\alpha_1}{\beta} \qquad o \qquad \frac{\alpha_1 \ \wedge \ \alpha_2}{\beta}$$

donde α_1 y α_2 son los <u>antecedentes</u> (condiciones bajo las cuales la regla de inferencia es aplicable) y ß es el <u>consecuente</u> (expresión a ser deducida). Cada antecedente es o una especificación de un comando (previamente demostrada) o una expresión del lenguaje de especificación que se debe demostrar separadamente como un lema.

Se usarán como reglas de inferencia las mismas usadas por Hoare:

Reglas de Consecuencia

D1
$$\frac{\{\alpha\}\ C\ \{\beta\} \ \wedge \ (\beta \dashrightarrow \gamma)}{\{\alpha\}\ C\ \{\gamma\}}$$

D2
$$\frac{\{\gamma\}\ C\ \{\beta\} \ \wedge \ (\alpha \dashrightarrow \gamma)}{\{\alpha\}\ C\ \{\beta\}}$$

donde C es cualquiera de los tres comandos básicos ya mencionados.

La secuenciación de comandos es un método de composición básico en cualquier lenguaje de programación, y que en este artículo solo ha sido usado implicitamente, será denotada separando los comandos mediante un símbolo de "punto y coma" (;).

Regla de Composición

D3
$$\frac{\{\alpha\}\ C_1\ \{\alpha_1\} \ \wedge \ \{\alpha_1\}\ C_2\ \{\beta\}}{\{\alpha\}\ C_1;C_2\ \{\beta\}}$$

Los axiomas en que se basará cualquier deducción serán los siguientes:

A0 Axioma de invocación inicial

Si al invocar un predicado especial $p(\bar{x},\bar{y})$, mediante la consulta $?\text{-}p(\bar{X},\bar{z})$, vale $(\alpha|\bar{x}:\bar{X})$ entonces se llega a un estado que cumple $\alpha \wedge \bar{y}=\bar{y}_1 \wedge \bar{x}=\bar{X}$. O sea

$$\{(\alpha|\bar{x}:\bar{X})\}\ \text{"invocación inicial de } ?\text{-}p(\bar{X},\bar{z})\text{"}\ \{\alpha \wedge \bar{y}=\bar{y}_1 \wedge \bar{x}=\bar{X}\}$$

A1 Axioma de Resolución

Si durante la ejecución de un procedimiento lógico p, corresponde realizar la resolución de ún átomo con identificador p con una cláusula C de la forma

$$p(\bar{u},\bar{v}) :- Q_1,\ldots Q_m$$

(para la cual la secuencia $Q_1,\ldots Q_m$ puede ser vacía), entonces

si dicho estado cumple el predicado α y además existe una sustitución Θ y un predicado ß tal que

(1) α ---> ("$\bar{u}$ y val($\bar{x}$) unifican via Θ como unificador más general")

y

(2) $(\alpha \wedge \bar{y}_1 = \text{val}(\Theta\bar{v}))$ ---> ß

después de realizar la resolución se llega a un estado para el cual vale ß. Es decir

$$\{\alpha\}\ \text{"resolución de objetivo con cláusula C"}\ \{ß\}$$

A2 Axioma de Procedimiento Externo

Si durante la ejecución de un procedimiento p de un programa lógico PL, se está en un estado para el cual un átomo con identificador r distinto de p, que corresponde a un procedimiento de PL distinto de p, es el primero en la lista de objetivos, y cumple con la especificación

$$\{\alpha\}\ r(\bar{m},\bar{n})\ \{ß\}$$

entonces, si dicho estado cumple $\mathcal{J}$ y $(\alpha|\bar{m}:\bar{M})$, (donde $\mathcal{J}$ es un predicado en el cual no ocurren las variables en $\bar{z}$, excepto talvez en igualdades de variables libres) una vez concluida la llamada al procedimiento r, se llega a un estado para el cual vale $\mathcal{J}$ y $(ß|\bar{m},\bar{n}:\bar{M},\bar{z})$. Es decir

$$\{\mathcal{J} \wedge (\alpha|\bar{m}:\bar{M})\}$$
$$\text{"llamada de procedimiento externo } r(\bar{M},\bar{z})\text{"}$$
$$\{\mathcal{J} \wedge (ß|\bar{m},\bar{n}:\bar{M},\bar{z})\}$$

A3 Axioma de invocación recursiva

Si durante la ejecución de un procedimiento lógico p, un átomo $p(\bar{M},\bar{w})$ que no corresponde a la invocación inicial del procedimiento P es el primero en la lista de objetivos, entonces si α vale y $(\alpha$ ---> $(ß|\bar{x},\bar{y}_1:\bar{M},\bar{w}))$ es cierto para un cierto predicado ß, se tiene que después de la invocación se llega a un estado

que cumple ß. Es decir

$$\{\alpha\} \text{ "invocación recursiva de la regla terminal RT" } \{ß\}$$

A4 Axioma de Terminación

Si durante la ejecución del procedimiento lógico $p(\bar{x},\bar{y})$ se llega a un estado para el cual, la lista de objetivos es vacía, y vale el predicado $(\alpha|\bar{z}:\bar{y})$ entonces, la ejecución de dicho procedimiento termina en un estado que cumple α :

$$\{(\alpha|\bar{z}:\bar{y})\} \text{ "terminación" } \{\alpha\}$$

Con estos axiomas y reglas de inferencia es posible demostrar la corrección de procedimientos lógicos especiales con respecto a una especificación dada; sin embargo, el nivel de detalle de estas deducciones resultaría poco claro y tedioso. Con el propósito de agilizar y hacer más intuitivo este proceso deductivo se presentará un axioma adicional, deducible de los anteriores pero útil para calcular corrección de aserciones sobre reglas recursivas terminales que no invocan procedimientos externos, y un teorema cuya aplicación permitirá demostrar la corrección de un procedimiento lógico especial con respecto a una especificación.

A5 Axioma adicional

Si durante la ejecución de un procedimiento lógico p, se está en un estado inmediatamente anterior a la resolución del primer átomo del objetivo actual con una regla de la forma

$$p(\bar{u},\bar{v}) :- p(\bar{t},\bar{w})$$

es decir una regla recursiva terminal que no incluye átomos correspondientes a procedimientos diferentes a p, entonces

si dicho estado satisface el predicado α, θ es el unificador más general para $\bar{u}$ y val($\bar{x}$) y existe un predicado ß tal que

$$(\alpha \wedge \bar{y}_1 = \bar{v}) \longrightarrow (ß|\bar{x},\bar{y}_1 : \theta\bar{t},\bar{w})$$

después de realizadas la resolución y la invocación recursiva se llega a un estado que cumple ß. Es decir

$$\{\alpha\}\ p(\bar{u},\bar{v}) :- p(\bar{t},\bar{w})\ \{ß\}$$

claramente este axioma combina los efectos estipulados por los axiomas A1 y A3.

Teorema 6.1 Suponga que en un programa lógico P, $p(\bar{x},\bar{y})$ es un procedimiento lógico especial con cláusulas no recursivas $CNP_1,\ldots,CNR_e$ y clausulas recursivas terminales $CR_1,\ldots,CR_f$ cuyo cuerpo responde a la siguientes especificaciones:

$\{\gamma_1 \wedge \mu\}$ CNR_1 $\{(\beta|\bar{x}:\bar{X})\}$

$\vdots$

$\{\gamma_e \wedge \mu\}$ CNR_e $\{(\beta|\bar{x}:\bar{X})\}$

$\{\varepsilon_1 \wedge \mu\}$ CR_1 $\{\mu\}$

$\vdots$

$\{\varepsilon_f \wedge \mu\}$ CR_f $\{\mu\}$

para ciertos predicados lógicos $\mu, \gamma_1, \ldots, \gamma_e, \varepsilon_1, \ldots, \varepsilon_f$ y si además

(1) $(\alpha \wedge \bar{x}=\bar{X} \wedge \bar{y}=\bar{y}_1) \Longrightarrow \mu$

(2) $\mu \Longrightarrow (\bigvee_{i=1}^{e} \gamma_i \circ \bigvee_{j=1}^{f} \varepsilon_j)$

(3) $\{\alpha\}$ $p(\bar{x},\bar{y})$ $\{T\}$ (T es el predicado universal)

entonces $\{(\alpha|\bar{x}:\bar{X})\}$ $(?- p(\bar{X},\bar{z}))/P$ $\{(\beta|\bar{x},\bar{y}:\bar{X},\bar{z})\}$

vale para toda sucesión de términos explícitos $\bar{X}$ y por tanto

$$\{\alpha\}\ p(\bar{x},\bar{y})/P\ \{\beta\}$$

μ se llama un <u>invariante</u> del procedimiento especial $p(\bar{x},\bar{y})$.

<u>Demostración</u>:

Del axioma A0 se obtiene

$$\{(\alpha|\bar{x}:\bar{X})\}\ \text{"invocación inicial"}\ \{\alpha \wedge \bar{y}=\bar{y}_1 \wedge \bar{x}=\bar{X}\}$$

entonces por (1) y la regla D1 se tiene

$$\{(\alpha|\bar{x}:\bar{X})\}\ \text{"invocación inicial de ?- } p(\bar{X},\bar{z})\text{"}\ \{\mu\}.$$

Ahora bien, del hecho anterior y del axioma (2) se deduce que exactamente después de la invocación inicial se presenta una de las siguientes dos situaciones:

(a) valen μ y algún ε_j, y el objetivo puede resolverse con una regla recursiva no terminal CR_j;

(b) valen μ y algún γ_i y el objetivo se resuelve con la regla no recursiva CNR_i.

En el primer caso (situación (a)) despues de ejecutarse las llamadas de procedimiento en CR_j se llega a un estado para el cual vale μ (de acuerdo a la especificación de CR_j); y entonces, debido a (2) uno de los predicados γ_i o ε_j vale, volviendo de nuevo a presentarse una de las situaciones (a) o (b). La hipótesis (3) garantiza que la ejecución del procedimiento termina normalmente, y por tanto eventualmente se llegará a un estado para el cual vale uno de los predicados $\gamma_1, \ldots, \gamma_e$ caso que corresponde a la situación (b).

En el segundo caso, después de ejecutarse las llamadas de procedimientos en en la cláusula CNR_i, se llega a un estado para el cual, la lista de objetivos es vacía y vale $(\beta|\bar{x}:\bar{X})$ por axioma A4 (puesto que $((\beta|\bar{x},\bar{y}:\bar{X},\bar{z})|\bar{z}:\bar{y}) = (\beta|\bar{x}:\bar{X})$); la ejecución del procedimiento termina entonces en un estado que cumple $(\beta|\bar{x},\bar{y}:\bar{X},\bar{z})$. Como $\bar{X}$ es una lista de términos explícitos arbitrarios

$$\{\alpha'\}\ p(\bar{x},\bar{y})/P\ \{\beta\}$$

vale. ∎

7. Ejemplos ilustrativos y aplicaciones.

En esta sección se presentarán dos ejemplos que muestran como desarrollar el cálculo de corrección presentado en este artículo de una manera ágil y efectiva.

Primeramente se mostrará la corrección del procedimiento lógico concat estudiado ya en los ejemplos 3.1, 4.1 y 5.1.

Ejemplo 7.1

El cuerpo del procedimiento concat puede anotarse así:

C1 $\{\mu \wedge (x_1=[])\}$ concat([],u,u) $\{y=X_1 * X_2\}$

C2 $\{\mu \wedge (x_1 \neq [])\}$ concat([p|t],u,[p|w]):- concat(t,u,w) $\{\mu\}$

siendo

$\mu \equiv [x_1, x_2, X_1, X_2$ son listas $\wedge$ $x_2=X_2 \wedge$

$(\exists l$: l es lista $\wedge$ $l * x_1 = X_1 \wedge y = l * y_1)]$

Usando el cálculo desarrollado en la sección 6 y con ayuda del teorema allí demostrado se probará que

$\{\alpha$: x_1, x_2 son listas$\}$ concat(x_1, x_2, y) $\{\beta$: $y = x_1 * x_2\}$

Dentro de la terminología de dicho teorema se tienen las siguientes correspondencias:

e=1 y $\gamma_1 \equiv (x=[])$, f=1 y $\varepsilon_1 \equiv (x \neq [])$ también

(1) corresponde a

$(x_1, x_2$ son listas $\wedge$ $x_1=X_1 \wedge x_2=X_2 \wedge y=y_1) \Longrightarrow \mu$
y su validez es clara.

(2) corresponde a

$[x_1, x_2, X_1, X_2$ son listas $\wedge$ $(\exists l$: l lista $\wedge$ $l * x_1 = X_1 \wedge y = l * y_1)] \wedge$

$\wedge$ $(x_2=X_2) \Longrightarrow (x_1=[]$ o $x_1 \neq [])$ que también vale.

(3) La terminación del procedimiento es intuitivamente clara pues el número de inferencias realizadas corresponde a la longitud de la lista X_1.

Se define ahora $\alpha_1 \equiv \mu \wedge (x_1=[])$ y $\alpha_2 \equiv \mu \wedge (x_1 \neq [])$ para mostrar que el código está bien anotado conforme al teorema 1.

Prueba de C1: (Se usa axioma 1) Suponga que α_1 vale. Sea θ la sustitución $\{(u, X_2)\}$ entonces val(x_1), val(x_2) (es decir [], X_2) unifican con $\bar{u}$ (ó sea [],u), con θ como unificador más general además

además

$$\alpha_1 \wedge y_1 = \theta u \equiv \mu \wedge x_1 = [] \wedge y_1 = X_2$$

$$\equiv (\exists l{:}l \text{ es lista} \wedge y = l * y_1 \wedge l * x_1 = X_1) \wedge x_2 = X_2 \wedge x_1 = [] \wedge y_1 = X_2$$

$$\Rightarrow y = X_1 * X_2 \quad (\text{porque } l = X_1)$$

y la prueba sigue del axioma A1

Prueba de C2: Suponga que α_2 vale, como [p|t] y u unifican respectivamente con val(x_1) y val(x_2) mediante la sustitución

$$\theta \equiv \{(p, car(x_1)),\ (t, cdr(x_1)),\ (u, X_2)\}$$

entonces por el axioma A5 basta mostrar que

$$\alpha_2 \wedge y_1 = \theta[p|w] \Rightarrow \mu | x_1, x_2, y_1 : \theta t, \theta u, w)$$ es decir

$[x_1, x_2, X_1, X_2$ son listas $\wedge\ x_2 = X_2 \wedge y_1 = [car(x_1)|w] \wedge x_1 \neq [] \wedge$

$(\exists l{:}l$ es lista $\wedge\ l * x_1 = X_1 \wedge y = l * y_1)] \Rightarrow [cdr(x_1), X_1, X_2$ son

listas $\wedge\ X_2 = X_2 \wedge (\exists l_1{:}l_1$ es lista $\wedge\ l_1 * cdr(x_1) = X_1 \wedge y = l_1 * w)]$

que se verifica facilmente.

Sobre este ejemplo se puede anotar lo siguiente:

(1) Aunque la semántica provista por la lógica clausal constituye en este caso, una explicación clara y natural del procedimiento en términos recursivos, no deja de ser interesante la explicación puramente iterativa que se obtiene con la interpretación en términos de transición de estados aquí desarrollada.

(2) Del cálculo detallado de corrección que se acaba de mostrar, es posible extraer una explicación iterativa más intuitiva y manejable:

Sabiendo que las reglas del procedimiento "concat" cumplen las siguientes especificaciones

C1 $\{\mu \wedge (x_1 = [])\}$ concat([],u,u) $\{y = X_1 * X_2\}$

C2 $\{\mu \wedge (x_1 \neq [])\}$ concat([p|t],u,[p|w]) :- concat(t,u,w) $\{\mu\}$

Se puede concluir que concat cumple:

$\{\alpha{:}x_1, x_2$ son listas$\}$ concat(x_1, x_2, y) $\{\beta : y = x_1 * x_2\}$

puesto que el resultado del teorema 6.1 vale si se cumplen las siguientes condiciones:

i) De la precondición α de concat debe poderse deducir que el invariante μ vale. Lo cual es evidente.

ii) Si el invariante μ y la condición x_1=[] valen entonces el procedimiento termina su ejecución mediante la aplicación del hecho concat([],u,u) cumpliendo con la postcondición $y=X_1*X_2$. Esto se deduce de que el efecto de la unificación con dicho hecho presupone que x_1=[] y que y_1 tome el valor de x_2.

iii) Si en algún estado de la ejecución del procedimiento valen el invariante μ y la condición x_1 es una lista diferente de [], entonces la regla en C2 es aplicable lo que hace que tanto x_1 como y_1 se actualicen respectivamente con los valores obtenidos de la aplicación de la funciónes cdr y cons (con car(x_1)) sobre sus valores previos. Las alteraciones del ambiente que se acaban de describir, hacen que después de la aplicación de la regla en cuestión valga de nuevo el invariante μ, lo que es facilmente verificable.

iv) La terminación del procedimiento está garantizada por los efectos producidos por la regla en C2 sobre el parámetro x_1, que hacen que su valor sea eventualmente igual a la lista vacía ([]).

A continuación se presenta un procedimiento para el cual la explicación de su corrección en términos de la semántica de la lógica clausal no es evidente, debido a su naturaleza iterativa. En contraste la interpretación en términos de transición de estados que propone este artículo da una explicación clara de su corrección, si se usa el modo informal de presentación que se ilustró al final del ejemplo anterior.

Ejemplo 7.2 Se desea demostrar que el procedimiento reverse(x,y) perteneciente al siguiente programa lógico

```
L:  reverse(u,v):- rev(u,[],v).
    rev([],u,u).
    rev([u1,u2],u3,v):- rev(u2,[u1|u3],v).
```

cumple la siguiente especificación:

{α: x es lista} reverse(x,y)/L {ß: $y=x^R$ (versión invertida de x)}

Basta probar que para todo termino explicito X y toda variable z se cumple

(*) {x es lista . x=X} ?-reverse(X,z)/L {$z=X^R$}

suponga entonces, que la siguiente especificación del procedimiento rev vale

(=) {α1: x_1,x_2 listas} rev(x_1,x_2,y)/L {ß1: $y=x_1^R*x_2$}

como el cuerpo del procedimiento rev consta de una sola regla, por el axioma A0 la prueba de (*) se reduce entonces a mostrar

(**) {x es lista ∧ x=X ∧ y=y_1} reverse(u,v):- rev(u,[],v) {y=X_R}

Para ello considere la sustitución θ = {(u,X)} para la cual u y val(x) unifican con como unificador más general. El axioma A1 garantiza que el estado en que se invoca a rev como procedimiento externo cumple el predicado

u es lista ∧ (y=y_1 ∧ y_1=v ∧ u=X)

luego por el axioma A2 después de realizada la invocación rev(u,[],v) vale

v=u^R ∧ (y=y_1 ∧ y_1=v ∧ u=X) de donde se deduce que y = X^R con lo que (**) queda demostrado.

Para demostrar la suposición hecha en (¤) se anotarán las reglas del cuerpo de rev de la siguiente forma

B1 {ξ : x_1=[] ∧ μ} rev([],u,u).{y=X_1^R*X_2}

B2 {ξ:x_1 es lista≠[]∧ }rev([u1|u2],u3,v):- rev(u2,[u1|u3],v){μ}

donde μ ≡ [x_1, x_2 son listas ∧ (x_2^R*x_1 = x_2^R*X_1) ∧ y=y_1]

Ahora bien, por teorema 6.1 es suficiente verificar las siguientes condiciones:

(i) La precondición α1 , y=y_1 y el hecho de que x_1=X_1 y x_2=X_2 son ciertas debe implicar μ , lo cual es evidente.

(ii) Si el invariante μ y la condición x_1=[] valen, el procedimiento termina su ejecución con la aplicación del hecho rev([],u,u) cumpliendo con la postcondición y=X_2^R*X_1. Esto se infiere de que la resolución con tal hecho presupone x_1=[] y que y_1 tome el valor de x_2 (que unifica con u) y por tanto se obtiene

x_2^R*[] = x_2^R*X_1 ∧ y_1=x_2 ∧ y=y_1 de donde se deduce que

y^R = X_2^R*X_1 y así y = X_1^R*X_2.

(iii) Si en algún estado de ejecución del procedimiento valen el invariante μ y la condición x_1 es lista diferente de [] entonces la regla en B2 es aplicable lo que ocasiona que tanto x_1 como x_2 se actualicen respectivamente con los valores $cdr(x_1)^2$ cons(car(x_1),x_2). Estas alteraciones preservan la validez del invariante μ después de la aplicación de la regla.

(iv) La terminación del procedimiento está garantizada por los efectos producidos por la regla B2 sobre el parámetro x_1 que hacen que su valor sea eventualmente igual a [].

Conclusiones.

La programación lógica fué concebida por Kowalski como un lenguaje de programación con una semántica natural y auto descriptiva. Sin embargo, PROLOG el lenguaje más popular que pretende implantar estas ideas, dista mucho de poseer las cualidades propuestas por Kowalski, en parte debido a los comandos y predicados adicionales que se le añadieron con el fin de hacerlo más eficiente y práctico [CM 84], pero también debido al estilo usado por los programadores profesionales, con el fin de escribir programas eficientes. Como ya se indicó en la introducción de este artículo la búsqueda de eficiencia se traduce en perdida de claridad para el programa, imponiendose de este modo, la necesidad del desarrollo de una noción precisa de corrección y de un cálculo para ésta.

En este trabajo se intentó dar un primer paso en esa dirección definiendo la noción de corrección de un programa lógico en términos de una interpretación de su ejecución mediante un esquema de transición de estados. La explicación de dicha interpretación no resultó del todo natural ni clara. Posteriormente se presentará una segunda versión de este artículo que supere dichas deficiencias.

Agradecimientos. El autor desea reconocer la influencia y consejo del profesor Rodrigo Cardoso en algunos aspectos del tratamiento de este artículo, sin embargo sobra anotar que los errores u omisiones que pudieran encontrarse son entera responsabilidad del primero.

Es importante también destacar la eficiente labor del Centro de Documentación del instituto CIFI de la facultad de Ingeniería de la universidad de Los Andes, en la persona de su director Jairo Mora sin cuya ayuda y soporte técnico en la consecución de la bibliografía necesaria este artículo nunca se habría escrito.

BIBLIOGRAFIA

[C73] Colmerauer A. et al, Un system de Comunication Homme-machine en Francaise. Rapport Groupe d'Intelligence Artificielle, Universite d'Aix Marseille, Luminy, 1973.

[CM84] Cloksin W.F., Mellish C.S., Programming in Prolog, 2nd ed. Springer Verlag, New York, 1984.

[D76] Dijkstra E.W., A Discipline of Programming, Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1976.

[H69] Hoare C.A.R., An Axiomatic Basis for Computer Programming. Comm. ACM 12 (10), pp 576-583, Octubre 1976.

[K74] Kowalski R.A., Predicate Logic as a Programming Language. Proc. IFIP 74, North Holland Publishing Co., Amsterdam, pp 569-574, 1974.

[K79] Kowalski R.A., Logic for Problem Solving, Elsevier North Holland, New york, 1979.

[R75] Roussel P., PROLOG: Manuel de Reference et d'Utilisation, Groupe d'Intelligence Artificielle, Universite d'Aix-Marseille, Luminy, Sept 1975.

[W77] Warren D.H.D., Pereira L.M., Pereira F., PROLOG The Language and its Implementation compared with LISP. Proc. Symp. on Artificial Intelligence and Programming Languages, SIGPLAN Notes, vol 12 No 8 and SIGART Newsletters No 64, pp 109-115, August 1977.

Una extensión de Lisp para la programación orientada a objetos.

Gustavo ARANGO
Grupo de Investigación ALIS
Departamento de Sistemas
Universidad de los Andes
Apartado Aereo 4496
Bogotá - Colombia

Resumen:

Se va a describir una extensión minima en Lisp para permitir la programación orientada a objetos. Estos últimos se representan como funciones y pueden ser utilizados de manera muy cómoda y flexible por los programadores. Se conservan las nociones de clases, operaciones y objetos y se permite la generación de transformaciones de objetos mediante el envio de mensajes. Las clases por su lado se pueden estructurar jerárquicamente y se maneja la noción de herencia de operaciones y mensajes.

1. MOTIVACIONES.

Al modelar situaciones de la vida real uno de los enfoques que se utiliza es basar la estructura de los programas en la estructura del sistema que se modela
se hace de esta manera la transformación:
objeto del sistema ---> objeto computacional

Para extender el modelo no se requieren cambios estrategicos en el programa solo se necesitan adiciones de análogos simbólicos de objetos o de acciones del sistema que se extrae del mundo real de forma natural

Una manera de hacerlo es mediante la programación orientada objeto (ver [GOL 83], [CHA 80]) que tiene como idea esencial la de llevar el estado de cada objeto (saber cómo y cuándo cambia manteniendo en todo momento su identidad)

El estado de un objeto se puede caracterizar mediante el uso de variables de estado que mantienen la información que permite obtener la historia del objeto o explicar su comportamiento.

En un sistema complejo los objetos son rara vez completamente independientes. Cada uno puede influenciar el estado de otros a través de interacciones que sirven para acoplar los cambios en las variables de estado de diferentes objetos.

Un enfoque interesante y fácil de manejar es agrupar las variables de estado en sub-sistemas cuando éstas están relacionadas de alguna manera. Para que el enfoque sirva y sea modular se deben descomponer en objetos que correspondan a modelos de los objetos reales del sistema.

2. UN EJEMPLO PARA ILUSTRAR LO QUE SE DESEA.

Supongamos que queremos representar la siguiente información como un tipo abstracto de datos:

- Se tiene una cuenta corriente que se abre originalmente con una cierta cantidad (esta cantidad representa la variable de estado que nos determina una cuenta particular; la llamaremos saldo).
- Sobre una cuenta se pueden efectuar dos operaciones: retiros (siempre y cuando no haya sobregiro) y depósitos. Cada una de estas operaciones modifica la variable saldo.

Se desea definir la clase de todas las cuentas y sus operaciones; la manera de generar instancias de dichas clase - ie: cuentas particulares - y una forma simple de manejar estas instancias que por medio de un mensaje identifique la operación a efectuar y produzca el efecto deseado.

Por ejemplo si cuenta_banco y caja_menor son dos instancias de la clase cuenta se desea poder afectarlas de la forma siguiente:

```
(cuenta_bancos deposito 10000)
(caja_menor retiro 10000)
```

Se espera que el resultado de dichas transacciones afecte el saldo de las cuentas respectivas y que depósito y retiro sean mensajes que desencadenen la apropiada operación de retiro o de depósito sobre la cuenta mencionada; mejor aún se desea que las cosas se hagan de una forma suficientemente modular para que una nueva operación y por consiguiente un nuevo mensaje sean totalmente independientes de las instancias y de las clases.

3. LO QUE SE NECESITA.

Para lograr lo anterior se va a trabajar con las siguientes funciones Lisp:

- (clase ...) permite introducir una clase

- (operacion ...) permite introducir una operación para una clase sin restricciones se pueden crear operaciones de un mismo nombre para diferentes clases

- (mensaje ...) permite introducir una lista de relaciones entre mensajes y operaciones entre clases
  sin restricciones se pueden crear varios mensajes para una misma operación

- (objeto ...) permite crear una instancia de una clase

-(subclase ...) permite generar una versión más sofisticada de alguna clase que hereda las operaciones de su superclase

4. LA IMPLEMENTACION EN LISP.

4.1. La creación de clases.

Se usa la forma general:

```
(clase NombreClase Variables_de_Estado)
```

para nuestro ejemplo de cuenta se tiene:
(clase cuenta saldo)

si la cuenta tuviera una clave de acceso se tendría:
(clase cuenta_clave saldo clave)
nótese que las variables de estado aparecen en sucesión

La definición de esta forma es:
```
(df clase *l:
  (prog ( )
    (put (car *l:) (cdr *l:) 'estado)))
```

Se utilisan las formas *df de dm* de algunas de las versiones de Lisp (ver [CHA 80])

Lo que se está haciendo es generar una lista de propiedades para el átomo identificado por NombreClase que asigna a su propiedad **estado** la lista de las variables de estado de la clase.

Para la definición de cuenta tenemos que la lista de propiedades del atomo **cuenta** es:
estado: (saldo)

4.2. La implementación de operaciones.

Para la definición de operaciones lo que se desea lograr se ilustra con un ejemplo.

La definición de un depósito es como sigue:
(setq saldo (+ saldo monto))
donde monto corresponde a la cantidad depositada

La forma de definición de la operación es:
(operacion deposito cuenta (monto) (setq saldo (+ saldo monto)))

y corresponde a la forma general:
(operacion NombreOperacion NombreClase Parametros Cuerpo)

Lo que se desea es generar la siguiente función que es utilizable por toda

instancia de la clase cuenta sin modificación ninguna:

```
(defun deposito:cuenta
  (lambda (::l monto)
     (prog (saldo)
        (setq saldo (nombre_excl ::l 'saldo))
        (return (set saldo (+ (eval saldo) monto))))))
```

Nótese el uso de la forma defun para definir las funciones en algunas versiones de Lisp - en particular para IQ-Lisp [IQ 83]

se puede ver que:

- se agrega a la función un nuevo parámetro ::l que en el momento del uso de una instancia cualquiera de la clase va a indicar como obtener las variables de estado de la instancia;

- se modifica el nombre de la función agregandole el identificador de la clase para poder así manejar varias operaciones con el mismo nombre para diferentes clases;

- el cuerpo de la función se modifica para permitir, por un doble direccionamiento, la obtención correcta de las variables de estado.

El texto que para cada operación define una función como la presentada anteriormente es:

```
(df operacion (*n* *t* *p* *c*)
  (prog (y)
     (setq y (get *t* 'estado))
     (eval (transforme_text (nombre_excl *n* *t*) y *p* (rep_l_atom y *c*))) ))
```

la función (nombre_excl Nombre_instancia Variable) genera un nombre particular para la variable que depende de la instancia; una posibilidad de implementación en Iq-Lisp es:

```
(de nombre_excl (n var)
   (intern (at (concat n (concat ":" var)))))
```

intern, concat, at son funciones de base de IQ-Lisp (ver [IQ 83])

Para ser eficientes con el manejo del espacio, la idea es generar un cuerpo único que pueda ser compartido por cada objeto creado. Se adiciona una serie de variables locales que se identifican con las variables de estado de cada objeto y se efectua un remplazo sobre el texto dado por el usuario para definir la operación.

El remplazo textual hace las siguientes tranformaciones sobre el cuerpo

- (**setq** Variable_global ...) <~ (**set** Variable_global ...)
- (**quote** Variable_global) <~ Variable_global
- Variable_global <~ (**eval** Variable_global)

estos remplazos permiten el manejo de las variables propias del objeto creado mediante un doble direccionamiento

Para la transformación del código de definición de una operación se utilisa la función:

(rep_l_atom Nombre_Objeto Globales Cuerpo)

que remplaza en Cuerpo toda ocurrencia de una variable que pertenezca a la lista Globales por un identificador exclusivo del objeto creado y al mismo tiempo ejecuta sobre el texto las transformaciones descritas anteriormente.

El texto de definición es:

```
(de rep_l_atom (lg text)
 (cond ((null text) nil)
       ((atom (car text))
          (if (memb (cadr text) lg)
             (cond ((eq (car text) 'setq)
                    (append (list 'set (cadr text))
                            (rep_l_atom lg (cddr text))))
                   ((eq (car text) 'quote)
                    (cadr text))
                   ((memb (car text) lg)
                    (cons (list 'eval (car text))
                          (rep_l_atom lg (cdr text))))
                   (t (cons (car text)
                            (rep_l_atom lg (cdr text)))))
             (cons (if (memb (car text) lg)
                       (list 'eval (car text))
                       (car text))
                   (rep_l_atom lg (cdr text))))
        (t
          (cons (rep_l_atom lg (car text))
                (rep_l_atom lg (cdr text))))))
```

Nótese que la función if no figura en todas las versiones de Lisp. Una manera de escribirla es:

```
(df if (l) |" (cond (@ (cadr l) @ (caddr l)) (t @(car (cdddr l)))
```

Nótese que se usan los símbolos de definición de macros. |" @ |@ que se encuentran en varias versiones Lisp [CHA 80] y [IQ 83]

La función **operacion** utiliza la función auxiliar transforme_text que se encarga del trabajo de construir el texto de la función genérica; su código es:

```
(de transforme_text (*n glob *p *c)
  |"(defun @ *n
     (lambda @(cons '::l *p)
       (prog @glob
             |@(mapcar
                 (lambda (x) |"(setq @x (nombre_excl ::l (quote @x))))
                 glob)
              (return @ *c)))))
```

4.3. La implementación de mensajes.

La definición de **mensaje** es bastante sencilla; su forma corresponde a.
(mensaje NombreClase ListaMensajeOperacion)

para la definición de los mensajes de la clase cuenta se puede hacer:

```
(mensaje cuenta (retiro . retiro) (deposito . deposito))
```

o

```
(mensaje cuenta (retiro . retiro))
(mensaje cuenta (deposito . deposito))
```

si se desea definir entrada como otro posible mensaje sinonimo de depósito basta agregar:

```
(mensaje cuenta (entrada . deposito))
```

La función que maneja las correspondencias entre mensajes y operaciones es:

```
(df mensaje :l:
  (prog (clase lista)
        (setq clase (car :l:)
        (setq lista (cdr :l:)
   ciclo (put 'mensaje (nombre_excl (cdar lista) clase)
                       (nombre_excl (caar lista) clase))
         (if (null (setq lista (cdr lista))) (return))
         (go ciclo)))
```

notese que los mensajes se guardan modificados con el nombre de su clase para de esta manera permitir que se puedan usar mensajes idénticos para operaciones distintas.

4.4. La implementación de instancias u objetos.

La impementación de una instancia es también algo complejo de realizar pero por el contrario es muy simple en su forma de uso.

Definir que cuenta_banco es una instancia que en Colombia requiere un saldo mínimo de 100000 pesos se hace con:
(objeto cuenta_banco cuenta 100000)

Se desea que la definición anterior produzca la siguiente función:

```
(defun cuenta_banco
  (nlambda *l:
   (prog (x)
     (setq x
        (ver_mensaje (car *l:) 'cuenta))
     (if
        (null x)
        (return "operacion desconocida")
        (return (apply x
                       (cons 'cuenta_banco
                             (cdr  *l:))))))))
```

Nótese el uso de la forma *nlambda* (ver [IQ 83]) para definir una función que en algunas versiones de Lisp es equivalente a *df* o a *fexpr* (ver [CHA 80])

la función ver_mensaje identifica el mensaje, obtiene la operación a ejecutar o produce un mensaje de error; en el caso de una identificación correcta, la operación se aplica sobre una lista de parámetros (esta lista incluye, además de los requeridos por la operación, el identificador del objeto para poder asociar correctamente las variables de estado).

Dado el alcance de Lisp, es necesario producir al mismo tiempo la inicialización de las variables de estado del objeto.

El texto que produce lo esbozado anteriormente es:

```
(dm objeto (l)
  |" (defun @(cadr l)
     (nlambda *l:
       (prog (x)
         @(bloque (lista_set (cadr l) (get (caddr l) 'estado) (cddar l))
                  |"(setq x (ver_mensaje (car *l:) (quote @(caddr l)))))
         (if (null x) (return "operacion desconocida"))
          (return (apply x (cons (quote @(cadr l)) (cdr *l:))))))))
```

la función lista_set se encarga de inicializar los valores de las variables globales con los argumentos del llamado; su forma general es:
(lista_set Nombre_Instancia Globales Actuales)

su definición es:

```
(de lista_set (n lg la)
  (if (not (null lg))
    (bloque (set (nombre_excl n (car lg)) (car la))
            (lista_set n (cdr lg) (cdr la)))))
```

nótese que la inicialización de las variables de estado se hace por intermedio de la función lista_set y es por lo tanto necesario hacerlo como un efecto de borde que se logra usando la función **bloque**.

La función **bloque** que permite una evaluación sequencial de una serie de expresiones Lisp se puede escribir usando la definicion de macros df:

```
(df bloque (l) |" (cond (t |@ (cdr l))))
```

El código de la función ver_mensaje se presentará una vez que se explique cómo funcionan las subclases de una clase.

4.5. La implementación de subclases.

Un tipo especial de cuenta es aquella que permite un saldo en rojo. Esta cuenta funciona de la misma manera que una cuenta normal para el caso de un depósito y tiene una forma propia para los retiros. Sus variables de estado son además del saldo, un límite de sobregiro que se le concede al dueño de la cuenta.
Para este caso especial es muy útil definir la clase cuenta_rojo como una subclase de cuenta.

La forma de hacerlo corresponde a:
(subclase cuenta cuenta_rojo rojo)

que corresponde a la forma general:
(subclase NombreSuperclase NombreClase Variables_de_Estado)

Como se aprecia en el ejemplo, las variables de estado incluídas son solamente las adicionales a las de la superclase (por eso se excluye saldo)

La función **subclase** produce el mismo efecto que la función **clase** definiendo como variables de estado las de la clase sumadas a las de la superclase y creando para la clase una nueva propiedad **superclase** en la que se indica la clase que la origina.

El texto de esta función es:

```
(df subclase *l:
  (prog( )
    (put (cadr *l:) (append (get (car *l:) 'estado) (cddr *l:)) 'estado)
    (put (cadr *l:) (car *l:) 'superclase)))
```

nótese que el orden de los argumentos de append es importante si se desea que las operaciones que se heredan de la superclase funcionen correctamente al usar las variables de estado

El ejemplo anterior genera la siguiente lista de propiedades para el atomo **cuenta_rojo**:

estado: (saldo rojo)
superclase: cuenta

Para definir operaciones y mensajes de una subclase se procede de la misma manera que para la clase. A continuación se presenta la definición de un retiro y su mensaje correspondiente para cuenta_roja.

```
(operacion retiro cuenta_roja (monto)
  (if (>= (- saldo monto) (- 0 rojo))
      (bloque (setq saldo (- saldo monto)) saldo)
      "se sobrepasa el saldo rojo autorizado"))

(mensaje cuenta_rojo (retiro . retiro))
```

La función, que se encarga de escoger la operación a efectuar segun el mensaje, recorre la cadena de superclases hasta encontrar la primera operación que corresponda.

Su código es:

```
(de ver_mensaje
   (men clase)
        (if (null clase)
            nil
           (let
              ((op
                  (get 'mensaje (nombre_excl men clase))))
               (if op
                  op
                  (ver_mensaje men
                           (get clase 'superclase))))))
```

5. UN EJEMPLO DEL MODELO ORIENTADO OBJETO.

5.1. Presentación del ejemplo.

Un ejemplo de la utilización de las funciones anteriormente definidas es la definición de la clase cuenta con clave (cuenta_clave).
Estas cuentas tienen dos variables de estado: su saldo y el codigo de acceso a la cuenta. Las operaciones posibles sobre estas cuentas son el depósito, el retiro y el cambio de clave.
Cada una de estas operaciones maneja dos informaciones el monto de la transacción y el código de acceso para permitirla (en el caso de retiro o depósito) y el viejo y nuevo códigos para el cambio de la clave.

5.2. Descripción del ejemplo en términos de objetos y mensajes.

Las siguientes funciones definen completamente el objeto computacional.

```
(clase cuenta_clave saldo clave)


(operación retiro cuenta_clave (monto validacion)
  (cond ((not (eq clave validacion)) "clave de acceso invalida")
        ((>= (monto saldo)) "fondos insuficientes")
        (T (setq saldo (- saldo monto)) saldo)))

(operación deposito cuenta_clave (monto validacion)
  (if (not (eq clave validacion))
      "clave de acceso invalida"
      (bloque (setq saldo (+ saldo monto)) saldo)))

(operación clave cuenta_clave (validacion nueva_clave)
  (if (not (eq clave validacion))
      "clave de acceso invalida"
      (bloque (setq clave nueva_clave) nueva_clave)))


(mensaje cuenta_clave (retiro . retiro) (deposito . deposito)
                      (cambio . clave) (proteccion . clave))
```

Se define además una subclase de la anterior que no acepta modificaciones de la clave de acceso:

```
(subclase cuenta_clave cuenta_clave_fija)
(operacion clave cuenta_clave_fija nil "no se permite el cambio de clave")
(mensaje (cambio . clave) (proteccion . clave))
```

5.3. Ejemplo de utilización.

Al definir las instancias.

```
(objeto cuenta_suiza cuenta_clave 1000000 'pepe)
(objeto cuenta_fija cuenta_clave_fija 1000 'nada)
```

Se produce el "diálogo" siguiente:

```
(cuenta_suiza cambio 'pepe 'pastor)
pastor

(cuenta_suiza retiro 10000 'pepe)
"clave de acceso invalida"

(cuenta_suiza retiro 2000000 'pastor)
"fondos insuficientes"

(cuenta_fija cambio 'nada 'pepe)
"no se permite el cambio de clave"

(cuenta_suiza clave 'pastor 'perro)
"operacion desconocida"

(cuenta_suiza deposito 50000 'pastor)
1050000
```

6. CONCLUSIONES Y PERSPECTIVAS.

Se da el código en Lisp para comenzar a hacer programación orientada a objetos en este lenguaje y esperamos que esta implementación sea utilizada (no se incluye el texto de las macros que sirven para definir las formas de, df y dm porque este depende de las versiones de Lisp utilizadas y es por lo general muy sencillo)

Las expresiones que se usan para definir una clase, sus operaciones y la creación sus instancias es de manejo muy simple. Para efectuar cambios en los objetos creados solo se necesitan conocimientos de la sintaxis de Lisp y del campo de aplicación.

Nuestra implementación favorece la facilidad de uso de los objetos y por esta razón, la protección de los objetos y la verificación de una correcta utilización de los mensajes con los parámetros apropiados no se tienen en cuenta.
Las mejoras a considerar para nuevas versiones - además de lo citado

anteriormente - debe permitir:
- controlar las modificaciones de las operaciones de una clase,
- definir reglas y alternativas de herencia de operaciones y mensajes,
- permitir que una clase pertenezca a varias superclases, ...

Aunque el número de mejoras posibles queda abierto, es importante anotar que el núcleo presentado es suficiente para introducir la programacion orientada objeto y para producir aplicacionas interesantes.

BIBLIOGRAFIA

[ABF 85] ABELSON H., SUSSMAN G. J. y SUSSMAN J. "Structure and Interpretation of Computer Programs". McGraw_Hill. 1985.

[CHA 80] CHARNIAK E., RIESBECK C. y McDERMOTT D. "Artificial Intelligence Programming". Lawrence Erlbaum Associates. 1980

[GOL 83] GOLBERG A., y ROBSON D. "SMALLTALK-80. The language and its implementation". Addison-Wesley Publishing Company, 1983.

[IQ 83] IQ-Lisp Reference Manual. Integral Quality.

LOGICAS NO MONOTONICAS

Lic. Raul Carnota.

ESLAI (Escuela Superior Latinoamericana de Informática)
Casilla de Correo 3193.
CP 1000 - Buenos Aires - Argentina.

RESUMEN

Lo que se conoce como razonamiento no monotónico aparece normalmente en el llamado "razonamiento de sentido común" y por ende en casi todas las áreas de la Inteligencia Artificial, que pretenden construir sistemas que reflejen la actividad humana "inteligente".

La intención de este artículo es trazar un panorama de los diversos intentos existentes para darle una base formal a la inclusión en los sistemas de AI de facilidades nomonotónicas.

El presente trabajo se ha realizado en el marco del proyecto API ("Ambiente de Programación Inferencial"), dirigido por el Dr. Jorge Vidart y financiado por el Programa Nacional de Informática y Electrónica (537-001) SECYT-Argentina

1. INTRODUCCION.

Resulta generalmente reconocido por los investigadores en Inteligencia Artificial(IA), que una de las características del razonamiento de sentido común que la lógica standard no puede reflejar es su no monotonicidad.

El presente trabajo tiene por objeto realizar una síntesis descriptiva de los principales formalismos propuestos en los últimos años para el tratamiento del razonamiento nomonotónico.

El siguiente ejemplo, debido a D.Gabbay ,ilustra claramente este tipo de razonamiento . Supongamos que la oficina de Stuttgart de una compañia de turismo ,está esperando a un grupo de pasajeros que vienen de Nueva York via Paris. A las 12 hs. la información de que disponen es la siguiente:

a)El vuelo partió desde Nueva York

b)El aeropuerto de Paris ha sido cerrado imprevistamente

Ante la imposibilidad de comunicarse con Nueva York, y de acuerdo a las normas habituales de la compañia aérea,se asume que:

c)El capitán del vuelo, al enterarse que Paris está cerrado, desviará el vuelo al aeropuerto de Londres.

En consecuencia, se infiere que:

d)Los pasajeros llegarán esa noche a Stuttgart provenientes de Londres.

Sin embargo, a las 14hs. la comunicación con Nueva York se restablece, y se conoce que:

a´)El avión partió con un retraso de varias horas del aeropuerto de Nueva York

Ante la nueva evidencia, se asume que:

c´)Al enterarse a poco de partir que Paris esta cerrado, el vuelo retornará a Nueva York

Entonces la conclusión a la que llega la oficina de Stuttgart es:

d´)Los pasajeros no llegarán esa noche a Stuttgart via Londres.

Cuál fue el esquema del razonamiento efectuado?

A las 12hs. a partir de a) & b) y asumiendo c), se deduce d).

A las 14hs. a partir de a) & b) & a´) y asumiendo c´), se deduce d´) que es la negacióde d).

En consecuencia, al añadirse una premisa al conjunto inicial,una conclusión previa deja de ser válida, por lo que el conjunto de "teoremas" no crece al crecer el conjunto de "axiomas" .

Generalizando el ejemplo, cualquier representación de un dominio complejo, si pretende ser consistente, es naturalmente incompleta, lo que lleva a razonar en base a hipótesis, posteriormente descartables junto con las consecuencias que, gracias a ellas, pudieron derivarse.

Otra forma común de este tipo de razonamiento se relaciona con las excepciones y la "tipicidad" (caracteristicas prototipicas de un objeto). Cualquier intento de clasificar objetos de un dominio real por algún criterio, siempre significa una simplificación. Los animales no estan "construidos" de acuerdo a la clasificación de los zoólogos. Por el contrario, esta se guia por ciertas características "esenciales" (desde algún punto de vista), pero limitadas respecto al universo total de observación.

Por lo tanto las excepciones ocurren inevitablemente y no resulta posible establecer condiciones necesarias y suficientes para la pertenencia a una clase dada, por lo que tienen sentido las formulaciones de "tipicidad" ("típicamente los pájaros vuelan"), y, consecuentemente, la existencia de excepciones .

Resulta natural, entonces, que los sistemas de IA tiendan a implementar mecanismos de inferencia no monotónica.

Asi,por ejemplo, los lenguajes frame-based tienen implementados valores por defecto cuyo funcionamiento, suponiendo que el dominio de interés fuesen personas clientes de una empresa de la ciudad de Córdoba, podría describirse asi:
"Si x es un cliente y no hay determinado ningun valor para la variable y tal que Residencia(x)=y,entonces asuma y=Córdoba"
Aqui el "no hay determinado" se entiende, tanto como la inexistencia de una asercion explicita cuanto la falla en deducir el correspondiente valor de y.

Si se interpreta |-/- como "falla en deducir", el razonamiento anterior se escribiría asi:

(x/Personas) |-/- (3y/Ciudades) Residencia(x,y)

Residencia(x,Córdoba)

Mas en general,

(x1/T1) (x2/T2)...(xn/Tn) |-/-(3y/T) P(x1,x2,...,xn,y)

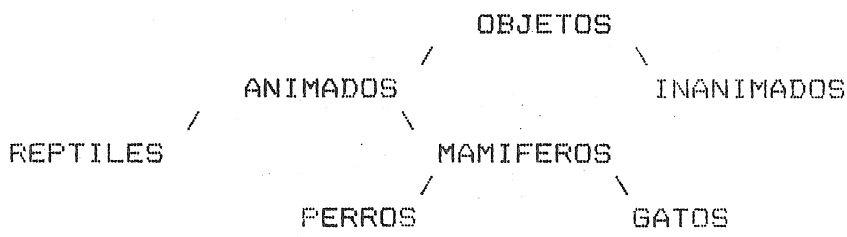
P(x1,x2,...,xn,<"default" para y>) (E1)

donde los Ti representan tipos a los que pertenecen las variables xi.

Si bien el sentido intuitivo del esquema (E1) es claro,no lo es el significado preciso de |-/-.

Un mecanismo conocido es la Asunción del Mundo Cerrado (CWA).

Supóngase una red semántica que describe los objetos del mundo :



Si Lassie es un perro, para deducir que es animado basta recorrer la red semántica hacia arriba, hasta hallar el nodo ANIMADOS. Si, en cambio, deseo probar que Lassie no es un reptil, recorro la red hacia arriba. Al no encontrar REPTILES arriba de PERROS se deduce "no es cierto que Lassie sea un reptil", a pesar de que esta información no aparece en la red. Lo que se ha aplicado en forma implícita es un esquema de razonamiento del tipo :

"Si Ud. falla en deducir que Lassie es un reptil (REPTIL(Lassie)), entonces concluya que no es cierto que Lassie es un reptil(-REPTIL(Lassie))".

Mas formalmente

$$\frac{\forall x \quad \vdash\!\!\not\!\!\!- P(x)}{\neg P(x)} \qquad \text{(E2)}$$

Los esquemas de regla (E1) y (E2) no son de primer orden, ya que $\vdash\!\!\not\!\!\!-$ es una metanoción.

La negación por falla de Prolog y otros lenguajes, no corresponde a la negación lógica, sino que es una variante del esquema (E2), es decir ,una regla por defecto.

En general, es cierto que la información negativa de un dominio es mucho mas voluminosa que la positiva (claramente son mas las ciudades a las que una compañia no vuela que sus destinos), razón por la cual resulta natural asumir la CWA.

En teoría ,la CWA supone "conocimiento perfecto" del mundo, es decir, tanto de la información positiva como de la negativa (aunque esta última no esté explícita en la KB). En la realidad, ante la presencia de información incompleta, como ocurre habitualmente, sostenemos la CWA y sacamos conclusiones en consecuencia ,conclusiones que dejan de ser válidas ante el agregado de nueva información. (P.ej. una nueva ciudad como destino). Esto hace del esquema (E2) una variante de razonamiento no monotónico.

Un problema clásico en la representación y manipulación del conocimiento es el denominado "frame problem", que tiene que ver con la representación de aquellos aspectos de un dominio cambiante en forma dinámica que permanecen invariantes bajo ciertas transformaciones del estado del mismo. Asi, por ejemplo, los objetos de una habitación no cambian de posición si se enciende la luz en la misma. Una representación en lógica de primer orden requiere explicitar estos invariantes. A estas explicitaciones se las llama "frame axioms". El frame axiom para el ejemplo anterior puede escribirse asi:

$$(x/\text{OBJETO})\ (s/\text{ESTADO})\ (l/\text{LLAVE-LUZ})\ (p/\text{POSICION})\quad \forall(x,p,s,l)\quad \text{POSICION}(x,p,s) \longrightarrow \text{POSICION}(x,p,\text{resultado}(\text{prender},l,s))$$

El problema es cómo resolver el frame problem sin necesidad de explicitar y manipular en la deducción todos los frame axioms. Una idea que presenta Reiter en [REI 78a] es que se necesita algun modo de afirmar que, en ausencia de información en contrario, un evento de cambio de estado preserva la verdad de una aserción.

$$\frac{(x1/T1)\ldots(xn/Tn)\ (s/\text{ESTADO})\ (f/\text{FUNCION-ACCION}) \quad \vdash\!\!\not\!\!\!- (\neg P(x1,\ldots,xn,f(x1,\ldots,xn,s))}{P(x1,\ldots,xn,f(x1,\ldots,xn,s))} \qquad \text{(E3)}$$

Como puede verse ,(E3) es tambien un molde de razonamiento no monotónico (dual en algun sentido al de la CWA).

Reiter muestra en [REI 78a] numerosos casos adicionales que requieren razonar con moldes similares.

Minsky señala que la lógica standard es monótona porque sus reglas de inferencia son en general del tipo "P es un teorema si Q1...Qn son teoremas", que el llama permisivas. Agregar axiomas sólo puede hacer que mas teoremas sean derivables y nunca invali-

dar uno previo.

En cambio, los moldes de razonamiento no monotónico (E1) a (E3), apelan a reglas de la forma "P es un teorema si Q1...Qn son no-teoremas". Si en algun momento posterior a su formulación ,algunos Qj se incorporan como premisas (o se hacen derivables de alguna nueva premisa),la deducción no es mas válida.

En la sección 2 se intenta fundamentar la necesidad de dar una base formal a los mecanismos que implementan la no monotonía. La sección tercera se destina a revisar críticamente algunos de los principales formalismos propuestos hasta la fecha, en tanto que en la sección 4 se discute lo presentado y en la sección 5 se extraen algunas conclusiones.

2.LOGICA Y ENFOQUES FORMALES.

Hasta ahora ha resultado mas fácil construir sistemas que razonen en forma no monotónica que justificar la corrección de dicho razonamiento.

La falta de claridad conceptual en las definiciones "prácticas" acarrea notables problemas. Un caso es la "herencia" de caracterÍsticas contradictorias como por ejemplo:

"El típico recién egresado de las Universidades no tiene empleo"

y

"El típico adulto tiene empleo"

Suponiendo que Alberto es un adulto recién egresado de la Universidad inferiremos que tiene empleo y que no lo tiene.

Tampoco parece aceptable aplicar la transitividad de los arcos IS-A en el caso de tratamientos con "tipicidad" (asunciones por defecto).Asi de las dos expresiones:

"El egresado típico de la Universidad es adulto"
"El adulto típico tiene empleo"

no parece intuitivamente correcto inferir

"El egresado típico de la Universidad tiene empleo"

Del mismo modo surgen inconvenientes al recorrer una jerarquía que representa alguna taxonomía con presencia de excepciones.

El marco natural para analizar formalmente los límites, confiabilidad, etc. de los mecanismos de representación del conocimiento es la lógica. Sin embargo, esta afirmación ha sido tradicionalmente polemizada por quienes sostienen la incapacidad de la lógica para reflejar el razonamiento de sentido común.

En los ultimos años varios autores,([LEV 84] ,[LEV 86], [NEW 81], [BRA 84]) han intentado esclarecer los aspectos confusos de dicha polémica. Por un lado se identificaba la lógica con la lógica clásica de 1er.orden, cuando existen, con distintos niveles de desarrollo, esquemas que extienden la lógica clásica para suplir precisamente las falencias que esta tiene con el objeto de captar adecuadamente el razonamiento humano en todas sus facetas (lógicas modales,temporales,etc), asi como lógicas alternativas (p. ej. las polivalentes).

De algun modo lo que Frege pretendió ,hace ya mas de un

siglo, fue echar los cimientos de lo que dio en llamar "ciencia del razonamiento puro" y si, en un primer momento histórico, este se identificó con el razonamiento deductivo "clásico", esto no agota el espectro del cálculo de ideas, que imaginaba Leibnitz.

Por otra parte, el rol de la lógica en representación del conocimiento merece ser examinado en los dos planos diferenciados que Newell distingue en [NEW 81] :el "nivel del conocimiento" y el "nivel simbólico". Esto es, la lógica como "teoría de verdad" que pretende analizar la correlación entre un formalismo de representación y el mundo que este aspira a reflejar, y la lógica en tanto formalismo de representación ella misma.

La lógica parece ser la herramienta adecuada para analizar el nivel del conocimiento. Dada una representación (red semántica, frame, estructura de datos para posiciones del ajedrez o cualquier otra), determinar exactamente qué conocimiento está siendo representado y caracterizarlo, requiere el uso de la lógica . Esto no significa exigir a todo lenguaje de KR que tenga un sistema de inferencias sound y completo, pero si conocer con precisión sus limitaciones .

En la misma línea, Nilsson en [NIL 80] utiliza el cálculo de predicados de primer orden como lenguaje común para presentar y discutir distintos formalismos de representación. Al mismo tiempo, presenta a la lógica (en conexión con los probadores de teoremas basados en resolución) como una representación particular, apta para ser aplicada en sistemas que resuelven problemas.

En este último rol, estamos en el "symbol level", donde la lógica es un representación posible (una familia de ellas) que resulta mas o menos adecuada, según el problema de que se trate.

Es en el contexto de esta necesidad de construir teorías de verdad como soporte de las diversas implementaciones de lenguajes de representación, que adquiere sentido la búsqueda de formalismos lógicos adecuados para las distintas formas de razonamiento no monotónico, los que deberán estar en condiciones de resolver los problemas descriptos en la sección 1, sin las dificultades expuestas al inicio de esta sección.

3. FORMALIZACIONES DEL RAZONAMIENTO NO MONOTONICO.

3.1 ASUNCION DEL MUNDO CERRADO (CWA).

Uno de los primeros intentos de desarrollo teórico de la no monotonicidad fue la formalización de la asunción del mundo cerrado en las bases de datos.

Reiter define asi, en [REI 78b], la clausura de una BD

$$CLAUSURA(BD) = BD \cup \{ \neg P(t) / BD \not\vdash P(t) \}$$

donde P son los símbolos de predicado n-arios en la BD y t las n-uplas de términos "ground" con símbolos de funcion que aparecen en la BD.

Asumir la CWA significa evaluar las preguntas contra CLAUSURA(BD) .

El problema es que la clausura conserva consistencia sólo si la BD esta constituida por cláusulas de Horn. Supóngase que

BD = {P \/ Q} entonces
CLAUSURA(BD) = {P\/Q,-P,-Q} lo que es inconsistente.

Incluso estando la BD constituida por átomos "ground", pero con constantes de Skolem (que representan un cuantificador existencial, es decir un individuo que existe ,pero sin identificación), se producen problemas, como se ve en el siguiente ejemplo:

BD = { CURSA(juan,c1), CURSA(pedro,c2), CURSA(w,c2) }
Dado que BD |≠ CURSA(w,c1) la CWA incluye en la clausura
- CURSA(w,c1)
lo que conduce a afirmar que
w ≠ juan
violando el sentido de w como constante indeterminada.

Clark en [CLA 78], realiza una formalización de la CWA pensando en la negación por falla de Prolog. El intuye que asumir la CWA equivale a afirmar que las condiciones suficientes, naturalmente expresadas en Prolog, son también necesarias, y provee un procedimiento de transformación de cláusulas (condiciones suficientes) en las fórmulas que representan las condiciones necesarias y suficientes.

En este formalismo, una pregunta contra la BD se evalúa contra la BD "completada" con las condiciones necesarias. Se trata de una propuesta muy dependiente del esquema de Prolog (sólo se define para cláusulas de Horn). Ademas pueden darse inconsistencias en casos especiales:
Si la BD contiene -P --> P, en la BD completada aparecerá
-P ≡ P.

3.2 LOGICA NO MONOTONICA.

Este enfoque fue propuesto originalmente por McDermott y Doyle en 1980 [MCD 80] y luego reformulado mas cuidadosamente por McDermott en 1982[MCD 82]. Se basa en la noción de "consistencia", que podría formularse asi:
"Si A puede ser asumida consistentemente, entonces, asúmala"
Para esto se introduce un operador M, cuyo sentido intuitivo seria "ser consistente":
MA
significa "A es consistente".

Una forma de entender la idea subyacente es pensar en una regla de inferencia que podría expresarse asi:
"Si no puede inferir -A , entonces MA "

El problema es definir el sentido formal de "no puede inferir". En este punto ,MacDermott no se refiere a ninguna nocion standard de derivación lógica sino a una relación de consecuencia lógica |~~ , tal que satisfaga
|~/~ (-A) entonces |~~ MA

McDermott, en su segundo trabajo, toma a M como un operador modal en alguna lógica modal de primer orden y construye la relación |~~ a partir de la noción de derivación de dicha lógica modal (T,S4, o S5).

Como la noción de consistencia se referencia a toda la KB, a la que la expresión MA pertenece, hay autorreferencia, por lo que la noción de "pertenencia al conjunto de teoremas" de una teoría no monotónica, surge de una construcción de punto fijo.

Si T es una teoría no monotónica, P es un punto fijo de T si

P = Th(T U {Mw / -w ∉ P})

que expresa la idea de que si la negación de w no es derivable, entonces asumo w.

Luego se define el conjunto de teoremas de la teoría no monotónica como la intersección de todos los puntos fijos.

Un ejemplo simple sería considerar la teoría nomonotónica T definida asi T={E /\ MC -->-D,
F /\ MD --> -C ,E ,E --> F} . De la primer fórmula se puede concluir -D, lo que hace que MD no sea válido en la segunda y por lo tanto no se pueda concluir -C de ella. Razonando en otro orden se puede asumir consistentemente D y concluir -C, pero no -D. En consecuencia hay dos puntos fijos :

P1=Th(T U {-D}) ; P2=Th(T U {-C})

Los teoremas de T serán los pertenecientes a la intersección de P1 y P2.

Si bien en la primera versión de 1980, Th simbolizaba el conjunto de consecuencias derivables (teoremas) en la lógica clásica, en la versión de 1982 la noción de derivabilidad es la de alguna lógica modal.

Un ejemplo clásico, el de los pájaros y el vuelo, se puede expresar en este formalismo del siguiente modo:

(∀x) (PAJARO(x) /\ M VUELA(x) --> VUELA(x)) (1)

Para MacDermott ,esta sentencia se puede leer " la mayoría de los pájaros vuela" ,resultando asi una forma de razonamiento por defecto. Esta interpretación no parece compadecerse con la asignación intuitiva a M del significado "es consistente". Mas bien, en la formulación:

"Es consistente afirmar que PI-PIO vuela si no puedo probar que no vuela",

parece decirse que los únicos pájaros que no pueden volar son aquellos de los que se puede inferir que no vuelan, en una línea análoga al formalismo de MacCarthy.

La lógica nomonotónica, uno de los primeros formalismos en este terreno, ha sido criticado desde distintos ángulos. Estos son algunos de los cuestionamientos:

-El uso de la lógica modal no parece intuitivamente justificado, ya que no surge clara la relación entre el sentido subyacente del operador M y la clásica semántica de "mundos posibles" de las lógicas modales.

-Se puede probar la consistencia del formalismo para la lógica S5 , pero en este caso resulta que S5 nomonotónica es igual a S5 monotónica (en tanto conjuntos de teoremas) y, por lo tanto, carente de interes.

-En cambio, en las lógicas T y S4, existen teorías no monotónicas inconsistentes (en el sentido de que MC pertenezca a la teoría y que de ella también pueda deducirse -C) ,siendo que las respectivas teorías monotónicas son consistentes.

3.3 LOGICA DEFAULT.

El enfoque de Reiter al presentar su lógica "default", ya se insinúa en [REI 78a]. Alli hace una presentación de las distintas situaciones en IA donde aparecen razonamientos de tipo no monotónico (asunciones y "defaults" en especial) y va unificando

la visión de los mismos a traves de formalizarlos como variantes de un mismo molde de inferencia, que es descripto, en cada caso, como una regla-esquema de inferencia (ver sec. 1).

Como conclusión, propone construir extensiones a la lógica clásica que incluyan reglas-esquema del tipo

$$\frac{\vdash\!/\!- B}{A}$$

Una instancia de esta regla ,por ejemplo, es la negación por falla usada en Prolog, que se expresa

$$\frac{\vdash\!/\!- B}{\neg B}$$

Sin embargo, ya en este primer trabajo, Reiter señala algunos inconvenientes del enfoque propuesto y que no estan claramente resueltos, entre ellos:

-Es facil construir teorías inconsistentes, aplicando la regla

$$\frac{\vdash\!/\!- A}{B}$$

a una KB donde vale -B.

-Si se tienen dos reglas

$$\frac{\vdash\!/\!- B}{A} \quad y \quad \frac{\vdash\!/\!- A}{B}$$

se derivará en la teoría A o B según el orden en que se hayan aplicado las reglas.

-Si no se asocia, de algun modo, lo deducido por las reglas por defecto a las asunciones que les dieron origen, se corre el riesgo de que permanezcan en la KB aun despues de que dichas asunciones hayan dejado de ser válidas.

El mismo Reiter, en [REI 80], desarrolla una lógica "default" en base a las ideas expuestas.

Las reglas-esquema son del tipo

$$\frac{A(x):B(x)}{C(x)}$$

que se entienden asi:

"Si A(x) es cierto y B(x) puede ser asumido consistentemente, entonces puede inferirse C(x)".

Una teoría "default" se define como <D,W> , donde

D:conjunto de reglas por defecto del tipo

$$\frac{A:B}{C} \text{ (sin variables libres)}$$

W:conjunto de sentencias de primer orden.

La idea es que W representa "verdades conocidas" en el dominio. Naturalmente, este conocimiento es incompleto y las reglas por defecto "mapean" estas teorías incompletas en extensiones mas completas.

La definición de dichas extensiones se hace con una construcción de punto fijo. Dado un conjunto de sentencias de primer orden S, se define el operador L del siguiente modo:

L(S) sera el menor conjunto que satisfaga
1-W $\subseteq$ L(S)
2-L(S) es cerrado bajo la consecuencia lógica de primer orden
3-Si $\frac{A:B}{C}$ esta en D y A E L(S) y -B E S ,entonces

C E L(S)

Las extensiones de la teoría <D,W> serán los puntos fijos del operador L, es decir, los R tal que L(R) = R .

Como ejemplo,si se tiene

$$W = \{ E, E \rightarrow F \} \qquad D = \{ \frac{E : C}{- D} ; \frac{F : D}{- C} \}$$

razonando como se explicó mas arriba , se llega a dos extensiones posibles
E1 = Th (W u {- D}) E2 = Th (W u {- C})
(donde Th(X) representa los teoremas generados a partir de X en 1er.orden)

En opinión de Reiter, cada extensión es un posible conjunto de creencias de un agente, algo asi como posibles "visiones"del mundo, compatibles con el conjunto original de verdades W.

A favor de la lógica "default", podemos decir que si las reglas son "normales", es decir, del tipo

$$\frac{A:B}{B}$$

existe una teoría de prueba, es decir, un procedimiento sintáctico de derivación, lo que permite determinar, dada una fórmula, su pertenencia o no a alguna extensión de una teoría si y si solo si la teoría de prueba lo afirma. Sin embargo, la condicion 3-que regimenta la consistencia de las asunciones por defecto, impide que las reglas sean efectivamente computables.

Una desventaja del enfoque, tal como se acepta en [REI 87], es que no se puede razonar acerca de las asunciones por defecto dentro de la lógica, ya que se presentan como reglas de inferencia.

Asi, por ejemplo, de la regla:
"Normalmente los canarios son amarillos" y la fórmula
"Si x es amarillo ,entonces x no es verde"
no es posible deducir la regla
"Normalmente, los canarios no son verdes",
lo que si parece factible dentro de la lógica no monotónica de McDermott .

Pese a sus fallas , la lógica "default" se ha usado en diversos trabajos tales como analizar situaciones presentes en redes semánticas con excepciones, razonamiento de diagnóstico , presuposiciones en lengua natural etc.

3.4 CIRCUMSCRIPCION.

La idea de este formalismo, propuesto por McCarthy y que ya aparece comentado en [MCC 77], es "completar" la información de una KB, asumiendo que los únicos objetos que satisfacen cierta

propiedad son aquellos que pueden generarse con los conocimientos que se poseen hasta el momento. Mas precisamente, se conocen algunos objetos en una clase dada y se tienen algunas vias de generar otros. Entonces, se "salta" a la conclusión de que todos los objetos de dicha clase se generan de ese modo. "Circumscribimos" la clase a aquellos objetos de la misma que sabemos como generar.

Si se conoce que a,b,y,c, tienen la propiedad P, y que si x la tiene, entonces f(x) la tiene ,el axioma de circumscripción para P se expresa con la siguiente sentencia-esquema :

F(a) /\ F(b) /\ F(c) /\ ∀ x (F(x)--> F(f(x)))
--> (1)
∀ x (F(x) --> P(x))

donde F es una "variable de predicado".

Lo que se esta diciendo, en un modo informal, es que P se define como la propiedad "mas chica" satisfecha por los objetos conocidos. Esto equivale a conjeturar que no hay otra manera de generar objetos que satisfagan P e induce una teoría no monotónica, ya que al crecer el número de premisas, pueden aparecer otros generadores de los {x/ P(x)}, con lo que (1) pierde validez.

Para que sean útiles las conclusiones extractables de circumscribir P, es necesario instanciar adecuadamente F.

En [REI 87], Reiter encuadra esta teoría entre los enfoques que denomina "basados en modelos mínimos" y que se caracterizan por que los teoremas de la teoría no monotónica son las sentencias verdaderas en todos los modelos "distinguidos" de una teoría lógica.

McCarthy, en su primer trabajo, considera como "distinguidos" a los modelos minimales respecto a una relación de orden entre modelos definida asi:

Asúmase a L como un lenguaje de primer orden. Supóngase P y Z tuplas de símbolos de predicado distintos tales que P y Z son disjuntas.

Dos modelos M1 y M2 son tales que

M1 ≤ M2 (segun P;Z) si

1)dom(M1) = dom(M2)
2)M1 y M2 interpretan todos los símbolos de función distintos de los de P y Z en forma idéntica
3)Para cada símbolo en P, su extensión en M1 es un subconjunto (no necesariamente propio) de su extensión en M2.

Si A(P;Z) es una sentencia de L que menciona a los predicados de P y de Z (y tal vez otros), los modelos distinguidos de interes para la teoría de McCarthy, son aquellos modelos de A(P;Z) minimales para el orden definido mas arriba. Las sentencias verdaderas en estos modelos minimales serán las derivaciones no monotónicas relevantes de A(P;Z).

En el enfoque actual de MacCarthy, se define la circumscripción de A(P;Z) con variable Z, como la sentencia de segundo orden

A(P;Z) /\ (∀ P';Z') -[A(P';Z') /\ P'< P] (2)

donde Q < R se define, dado un par de símbolos de predicado de la misma aridad, del siguiente modo:

(∀ x) (Q(x) --> R(x)) /\ - (∀ x) (R(x) --> Q(x))

y la relación entre tuplas de predicados se define como la conjunción de las relaciones Qi < Ri , supuesto que tengan dos a dos la misma aridad.

El axioma de circumscripción es la segunda parte de (2), que dice que las extensiones en A(P;Z) de los predicados P no pueden hacerse menores

" P es minimal en A con Z variando"

Las consecuencias no monótonas de interés que pueden obtenerse de A son las derivables de (2). Este enfoque mas sintáctico coincide (como sería de esperar) con el anterior. Es decir que las sentencias verdaderas en todos los modelos minimales de A(P;Z) coinciden con las derivables de (2).

El problema es que no resulta obvio como instanciar el axioma de circumscripción de modo de producir consecuencias interesantes.

McCarthy propone un principio uniforme para el molde "normalmente o típicamente tal y tal es el caso". Los predicados a minimizar son unos símbolos unarios ABi (por abnormal). Estos conforman la tupla P, mientras que en Z tenemos las propiedades respecto a las cuales se define anormalidad. Un ejemplo clásico es el siguiente:

(∀ x) OBJETO(x) /\ -AB1(x) --> -VUELA(x) (3)
(∀ x) PAJARO(x) --> OBJETO(x) /\ AB1(x) (4)
(∀ x) PAJARO(x) /\ -AB2(x) --> VUELA(x) (5)
(∀ x) PINGUINO(x) --> PAJARO(x) /\ AB2(x) /\ -VUELA(x) (6)

Si a la conjunción de (3) a (6) se la llama

A(AB1,AB2 ; VUELA)

se trata de minimizar (AB1,AB2) con VUELA variable. Esto significa intuitivamente hallar la menor cantidad de individuos anormales posibles, mas exactamente aquellos forzados a serlo por la teoría.

El axioma de circumscripción en este ejemplo es de la forma:

(∀ AB1' AB2' VUELA') -[A(AB1',AB2';VUELA') /\ AB1 < AB1' /\ AB2 < AB2'] (7)

Ahora se debería instanciar en (7) AB1',AB2',VUELA'.

Una instanciación intuitivamente adecuada puede ser la siguiente:

AB1'(x) ≡ PAJARO(x)
AB2'(x) ≡ PINGUINO(x)
VUELA'(x) ≡ PAJARO(x) /\ -PINGUINO(x)

Sustituyendo en (7) , la instancia resultante, junto a la sentencia original A(AB1,AB2;VUELA) constituída por (2) a (5) permite derivar :

(∀ x) AB1(x) ≡ PAJARO(x)
(∀ x) AB2(x) ≡ PINGUINO(x)

que afirma que las únicas cosas que son anormales respecto al no-vuelo son los pájaros y que los únicos pájaros anormales respecto al vuelo son los pinguinos.

Ahora es facil deducir consecuencias que no eran inferibles previamente ,concretamente:

(∀ x) OBJETO(x) /\ -PAJARO(x) --> -VUELA(x)
(∀ x) OBJETO(x) /\ -PINGUINO(x) --> VUELA(x)

Segun la opinión de varios autores (Reiter[REI 87],Levesque [LEV 86]), circumscripción parece ser el formalismo mas rico de todos los presentados en el terreno del razonamiento no monotónico. Tal vez esto se deba a que es el mas propenso al análisis matemático, de resultas de lo cual sus propiedades formales han sido extensamente estudiadas, probándose diversos resultados, incluso la provisión de una semántica para la negación en una clase amplia de programas Prolog (Lifschitz [LIF 86] desarrolla en varios aspectos la teoría de MacCarthy y sus aplicaciones).

Sin embargo la teoría de circumscripción presenta varios inconvenientes:

1-No es obvio en general como instanciar el axioma de circumscripción para que se produzcan consecuencias interesantes. Hay trabajos en desarrollo en este sentido.

2-Dado que las teorías circumscriptas son de segundo orden, sus fórmulas válidas no son, en general, recursivamente enumerables. Es notable que fenómenos del mismo tipo se producen en la lógica "default" y en la lógica no monotónica, al considerarse los no teoremas para definir la noción de "es consistente con".

3-Puede ocurrir que una teoría satisfacible, tenga un teoría circumscripta insatisfacible. Esto no ocurre si todas las sentencias de la teoría son universales y están en forma normal prenex.

3.5 LOGICA AUTOEPISTEMICA.

Retomando en forma crítica el desarrollo de McDermott y Doyle, Moore en [MOO 85] propone una lógica autoepistémica y realiza un desarrollo de la misma en el plano del cálculo proposicional.

Moore sugiere una distinción entre razonamiento por defecto y razonamiento autoepistémico.

Por razonamiento por defecto entiende la realización de inferencias plausibles, a partir de evidencias no categóricas y en ausencia de información en contrario. Si se sabe que PI-PIO es un pájaro, se tiene cierta evidencia de que PI-PIO puede volar, pero no es algo categórico. En ausencia de información opuesta, sin embargo, se sigue sosteniendo como conclusión que PI-PIO puede volar. El razonamiento por defecto, concluye Moore, no es por lo tanto una forma de realizar inferencias válidas.

En el formalismo de McDermott y Doyle se puede escribir:

$$\forall x\ (PAJARO(x) \wedge M(VUELA(x)) \rightarrow VUELA(x))$$

que dice que los únicos pájaros que no pueden volar, son aquellos para los que puede inferirse que no vuelan. En consecuencia, si se tiene una teoría cuyos únicos axiomas son la fórmula de arriba y una aserción que dice que PI-PIO es un pájaro, entonces debe ser cierto que PI-PIO puede volar (resulta una inferencia válida).

Segun Moore, este tipo de razonamiento es acerca de los propios conocimientos o creencias del agente que razona (por ej. la KB), y lo llama razonamiento autoepistémico. Para ilustrar las diferencias entre ambos, considera el ejemplo del hermano mayor : " Si yo creo que no tengo un hermano mayor, no es porque mis padres me lo informaron, o porque yo haya realizado una investigación cuidadosa al respecto. Simplemente creo que si

tuviese hermanos mayores lo debería saber; dado que que no se nada acerca de eventuales hermanos mayores, entonces concluyo que no tengo ninguno".

El discurso anterior es muy diferente al siguiente razonamiento por defecto: "los típicos egresados del MIT son hijos mayores; como yo soy un egresado del MIT, probablemente sea hijo mayor".

El razonamiento autoepistémico puro no sostiene conclusiones descartables. Sin embargo es también no monotónico, en el sentido de que es sensitivo al contexto (a la teoría en la que esta inmersa la sentencia en cuestión).

En la teoría cuyos únicos axiomas son:

(1) PAJARO(PI-PIO)

(2) ∀x(PAJARO(x) /\ M(VUELA(x)) --> VUELA(x))

una fórmula MP significa que P es consistente con la teoría no monotónica que contiene esos dos axiomas. En ese contexto, cabe esperar que VUELA(PI-PIO) sea un teorema de la teoría. Si luego se cambia la teoría, agregando:

(3) - VUELA(PI-PIO)

como axioma, el sentido de MP también cambia y significará que P es consistente con la teoría que contiene (1) a (3) como axiomas, en la cual no es mas esperable que VUELA(PI-PIO) sea un teorema.

En la lógica de Moore el operador primitivo es de creencia y puede definirse en términos del operador de consistencia de McDermott asi L = - M - (un agente racional, "cree" A si no es cierto que no(A) es consistente o, desde la noción de consistencia, una fórmula es consistente si su negación no es creída).

Una teoria autoepistémica es un conjunto de fórmulas lógicas interpretadas como las creencias totales de un agente.

Si consideramos la teoría cuyos axiomas son (1) y (2) pero cambiando M por - L -, el conjunto de creencias del agente podrá expandirse para contener

- L -(VUELA(PI-PIO)) y luego, por Modus Ponens,

VUELA(PI-PIO).

A diferencia de McDermott, Moore no se basa en ninguna axiomática modal, sino que construye el concepto de derivación no monotónica a partir de la lógica clasica, (en principio, el cálculo proposicional), y de modo tal que - LF es cierto siempre que F este fuera del conjunto de creencias del agente.

Una expansión estable T de un conjunto de premisas A se define como:

T=Th(A U {LW / W E T} U {-LW / W E T})

donde Th es el operador de clausura bajo la derivación lógica del cálculo proposicional.

Dadas las premisas en A, las teorias autoepistémicas (las "creencias" de la KB) son las expansiones estables de A. Pueden existir mas de una o ninguna expansión estable para un conjunto A. Por ejemplo, si A={- LP--> Q ,- LQ--> P}, una expansión estable de A contendrá Q y no P y otra expansión estable contendrá P y no Q. Del mismo modo, la premisa {- LP-->P} no tiene expansión estable alguna (cualquier intento lleva a contradicción). Lo que ocurre es que el operador L , como ya fue

expresado, se interpreta en el marco del conjunto total de creencias, por lo que su interpretación cambia segun las distintas maneras de completarse dicho conjunto.

Para dejar definido en forma única el conjunto de teoremas de una teoría, podría seguirse el camino de intersectar todas las expansiones estables. Para Moore esto sería razonable , si se piensa en lo que un observador externo debería saber del agente. Desde el punto de vista del agente, se trata de perspectivas alternativas que se le presentan.

Las expansiones estables son una construcción de punto fijo comparable con la de McDermott y Doyle.

Estos, en su primer trabajo realizan su construcción de punto fijo a partir de las premisas A y definen:

T= Th(A U {- LW / W E T})

mientras que Moore agrega al conjunto base

{LW / W E T}

que representa "introspección perfecta positiva" (del agente respecto a sus propias creencias). Moore critica a McDermott que sus agentes son omniscientes de todo aquello que no creen, pero pueden no saber nada de sus propias creencias.

En su segundo trabajo, McDermott define los puntos fijos en términos de las consecuencias modales del conjunto base. De este modo se introduce P |- LP como regla de inferencia y los puntos fijos generados a partir de A son todos expansiones estables de A. La inversa no es cierta, ya que no es equivalente introducir LP a partir de la regla de inferencia que a través del conjunto base de la construcción de punto fijo. Moore explica que en el esquema de McDermott al adoptar la lógica modal S5, se incluye el axioma-esquema LP --> P (todo lo creído es verdadero), por lo que la S5 no monotónica se reduce a la S5 monotónica.

Konolige, en [KON 87], detecta que la multiplicidad de expansiones estables generan ciertas incongruencias entre la teoría de Moore y las intuiciones acerca de la interpretación de las expansiones. Para salvarlas, define "expansiones fuertes" como expansiones estables minimales, donde el concepto de minimalidad se toma sobre las formulas "objetivas" (aquellas que no poseen el operador L), lo que limita las posibles asunciones del agente respecto del mundo.

Una caracterización formal de las expansiones fuertes se puede lograr en términos de una lógica modal KU45, intermedia entre S4 y S5, donde no vale LP-->P y si, en cambio, L (LP-->P) dentro de una construcción análoga a la de Moore.

Konolige también realiza una comparación entre la lógica autoepistémica y la "default" mediante una transformación que traduce reglas por defecto a sentencias de la lógica autoepistémica,

la regla $\frac{A : B}{C}$ se convierte en (LA /\ - L -B --> C)

y demuestra que hay una correspondencia exacta entre las extensiones de una teoría por defecto y las expansiones fuertes de una teoria autoepistémica.

Levesque [LEV 87] generaliza las expansiones estables de Moore al caso de primer orden y las caracteriza semánticamente en

términos de un segundo operador O tal que OW se lee "W es todo lo que el agente cree" y OW es cierto exactamente cuando todas las fórmulas creídas forman una expansión estable de (W).

3.6 OTRAS PROPUESTAS.

Una propuesta diferente en cuanto a la base lógica es la de Turner [TUR 84]. En la misma utiliza la lógica parcial (con dos valores de verdad mas un "indefinido" o "agujero"), partiendo de que la motivación principal del razonamiento no monotónico está dada por el conocimiento parcial o incompleto del mundo.

Un modelo parcial M para el lenguaje L, se define como

M = <D,F>, donde D es un dominio y F una asignación a cada símbolo de relación n-ario C de L de una función CM_n

$CM_n : D \dashrightarrow \langle V, F, U \rangle$

Turner considera crucial el concepto de "asunción plausible" y por ende, define "extensiones plausibles" del estado actual de conocimiento. Estas extensiones están ligadas a un orden $\sqsubset$ definido entre los modelos parciales, donde $M \sqsubset M'$ significa "M' es una extensión plausible de M". Para definir el orden $\sqsubset$, se define antes la relación de extensión natural : "Si M y M' son modelos parciales de L, M' es una extensión de M si para cada símbolo C de L, la función CM' en M' es una extensión de la respectiva función CM en M ".

Posteriormente se desarrollan algunas ideas para definir mejor las propiedades de la extensiones plausibles, con la expresa intención de dejar planteados algunos aportes para una semántica diferente a la modal para la lógica no monotónica de McDermott y Doyle.

Por su parte, Bossu y Siegel [BOS 85] proponen un nuevo método basado en la lógica de primer orden. Definen una relación $\models$ que llaman "subimplicación", a partir de un enfoque de modelos minimales próximo a la circumscripción de McCarthy.

La subimplicación se define a partir de una relación de orden entre interpretaciones, donde "I es inferior a J, si cada hecho verdadero en I es también verdadero en J". La diferencia entre los modelos minimales definidos segun este orden y los propuestos por McCarthy en [MCC 80], reside en que este último minimaliza sólo sobre algunos literales, mientras Bossu y Siegel lo hacen sobre todos ellos.

Pese a que no todo conjunto de formulas tiene modelo minimal, se demuestra que el subconjunto clausal si lo tiene.

La subimplicación tiene la propiedad de ser coherente. Sea P un conjunto consistente de fórmulas y q una fórmula. Se prueba que

si $P \models q$ entonces $P \not\models \neg q$

Además $\models$ satisface tres condiciones:

1-Si $P \models q$ entonces $P \models q$ (si q esta entre las consecuencias lógicamente derivables de la KB representada por P, la pregunta "es cierto q?" debe responderse por si).

2-$P \models q1 \wedge q2$ si y solo si $P \models q1$ y $P \models q2$.

3-Si q es positiva (no contiene la negación ni la implicación) vale que P]= q si y solo si P |= q (esto significa que la subimplicación es monótona en las fórmulas positivas).

Las dos primeras condiciones son naturales, mientras que la tercera es mas "subjetiva" en opinión de los propios autores.

La subimplicación definida del modo expresado es una relación no monotónica (si P esta incluído en P', el conjunto de las fórmulas subimplicadas por P no esta necesariamente incluído en el conjunto de fórmulas subimplicadas por P'). En particular se prueba que si P es vacío o está constituído solo de fórmulas negativas, entonces toda fórmula negativa está subimplicada por P (lo que es consistente con la CWA).

En base a la construcción de modelos minimales se define una función val[q,P] a valores {1, 0, 1/2}, de modo tal que, si P es una base de datos y q una pregunta respondible por si-no, se tiene por respuestas:

si , si P]= q (val[q,P]=1) ;
no , si P]= - q (val[q,P]=0) ;
indefinido , si P]≠ q y
P]≠ - q (val[q,P]=1/2).

Lo mas significativo de este desarrollo es que, bajo ciertas restricciones sobre las cláusulas, existe un mecanismo de decisión (descripto en el artículo y basado en una estrategia de prueba de teoremas por resolución llamada de saturación) para la prueba de P]=q y que esta prueba equivale a afirmar que q es cierto en todos los modelos mínimos de la teoría y por lo tanto también es deducible de la teoría por circumscripción.

La restricción consiste en que P debe estar compuesto de cláusulas cuyos únicos símbolos de función son las constantes y en las cuales cada variable que aparece en un literal positivo también aparece en un literal negativo, esto es:

-las cláusulas "ground", en particular la vacía.
-las cláusulas negativas.
-las cláusulas semánticamente equivalentes a una sentencia del tipo:

(∀ x1...xp) (l1 \/...\/ln --> l1'\/...\/lm')

donde los li y los lj' son literales positivos y donde cada variable xk aparece en al menos uno de los li.

En cuanto a las preguntas, la restricción es menos natural, ya que se les exige que su forma prenexa sea cuantificada universalmente.

4. DISCUSION.

Un elemento comun que se puede apreciar en varios formalismos propuestos es la apelación a cierto concepto de "minimalidad" (modelos mínimos, intersección de puntos fijos, extensiones minimales,etc).

Shoham en [SHO 86], propone una visión unificadora en base a dos perspectivas:

a)Priorizar las caracterizaciones semánticas por sobre las sintácticas, que son las predominantes en las lógicas "default", no monotónica y autoepistémica. La idea es fijar un lenguaje lógico

no necesariamente de primer orden, y luego proveerlo de una semántica adecuada a la relación de derivación que cada aplicación puede necesitar.
b)La relación de derivación en cuestión, deberá estar definida en términos de "verdad en todos los modelos minimales de la axiomatizacion dada" donde el criterio de minimalidad sería dependiente de la aplicación. Para Shoham, una axiomatización de un dominio de aplicación, junto con una caracterización de los modelos mínimos deseados, es una especificación suficiente para la relación de derivación no monotónica requerida.

Otra problema comun, parece sustentarse en que las descripciones de dominios razonablemente complejos son naturalmente incompletas, y las apelaciones a asunciones ,"default"s,etc. buscan "completarlas" llenando provisoriamente los "agujeros" con aserciones válidas, mientras no haya evidencia en contrario, lo que conduce a inferencias basadas en reglas "no permisivas" (P es teorema si Q1...Qn son no teoremas), una de cuyas instancias mas conocidas es:

si !-/-W infiera - W

En las teorías de primer orden, los no teoremas no son recursivamente enumerables (si lo fueran, los teoremas formarían un conjunto recursivo). Por lo tanto no existe un algoritmo que, dada una fórmula W, pueda establecer que no es teorema, aun en el caso de que esto sea asi.

Esto incide en que no existan lo que se denomina teorías de prueba, salvo para el caso proposicional o bajo restricciones(Bossu y Siegel).

Un ejemplo demostrativo de las dificultades que subsisten, lo suministran Hanks y McDermott en [HAN 86]. Alli investigan varias propuestas de solución del "frame problem", mostrando como enfoques naturales que usan algunos de los formalismos vistos en la seccion 3, pueden fallar. El caso-ejemplo es simple: en un estado s0 una persona esta viva, luego se carga un revólver, pasa cierto tiempo, y el revólver es disparado contra la persona. La cuestión es si la muerte de la persona puede deducirse no monotónicamente, sin el uso de frame axioms. La axiomatización usa el predicado AB propuesto por McCarthy, y un predicado T, donde T(f,s) denota que el hecho f es verdadero en el estado s.

```
T(vivo,s0)
(∀s) T(cargado,resulta(cargar,s))
(∀s)      T(cargado,s)      -->      AB(vivo,disparo,s)      /\
                                     T(muerto,resulta(disparo,s))
(∀f,e,s) T(f,s) /\ -AB(f,e,s) --> T(f,resulta(e,s))
```

En la última sentencia, se pretende obviar el uso de los frame axioms, mediante la afirmación de que, si el hecho f no es anormal cuando ocurre el evento e en el estado s, entonces f permanece cierto en el estado que resulta de que el evento e ocurra en el estado s.

Supongase que se desarrollan las siguientes situaciones: s0, s1=resulta(cargar,s0), s2=resulta(espera,s1), s3=resulta(disparo,s2).

Resultaria natural que, si se circumscriben los axiomas, minimizando AB con T variando, deberia poder derivarse T(muerto,s3).

Eso no ocurre, porque la teoría circumscripta tiene dos modelos mínimos en AB. En uno, AB(vivo,disparo,s2) es el único átomo AB verdadero. En este modelo T(muerto,s3) es verdad. Pero en el otro modelo minimal en AB, el único átomo AB verdadero es AB(cargado,espera,s1) (como si el revólver se descargase durante la espera) y por lo tanto es cierto T(vivo,s3).

Un resultado análogo ocurre usando lógica "default" y la regla-esquema

$$\frac{: \neg AB(f,e,s)}{\neg AB(f,e,s)}$$

ya que aparecen dos extensiones, una conteniendo T(muerto,s3) y la otra T(vivo,s3).

Si bien es cierto que existen otra formas de axiomatizar la misma situación que conducen a conclusiones correctas, la expuesta arriba es suficientemente natural como para sembrar la duda acerca de los formalismos en cuestión.

Es posible que las dificultades observadas en este caso, se deban a que en el planteo del "frame problem" debería hacerse hincapié en el rol especial del tiempo. Existen diversas propuestas que utilizan la noción de evento, intervalo,etc. (Kowalsky y Sergot, Allen, entre otros autores) y lo aplican a desarrollos para encarar el "persistence problem" . Tal vez la mas desarrollada sea el cálculo de eventos de Kowalsky y Sergot [KOW 86a,b], que utiliza la negación por falla de Prolog para encarar la no monotonicidad. Su inconveniente es que reposa en un mecanismo cuya semántica aun no esta suficientemente aclarada, asemejándose en todo caso mas a una implementación práctica que a una especificación formal.

Describir estos enfoques queda fuera del alcance de este trabajo.

Una crítica a las formalizaciones basadas en lógicas mas o menos convencionales , proviene de los enfoques estadísticos. Estos identifican tipicidad con frecuencia . Asi "Típicamente los pájaros vuelan", es interpretado como "La mayoría de los pájaros vuela", y esta lectura parece razonable en este caso. Sin embargo, si se piensa en el caso de una representación mediante una fotografía o diagrama, mas la asunción de mundo cerrado de que todo lo que no aparece en dicha representación, no está presente en el dominio representado, el rol de la estadística no aparece claro

Otra discusión gira en torno al hecho de que buena parte de los razonamientos no monótonos sirven para proveer explicaciones plausibles para algún estado del universo bajo consideración (este tipo de razonamiento se llama abductivo y el mejor ejemplo es el de diagnóstico, donde se busca una explicación para un conjunto de síntomas). Si bien una explicación puede verse desde el ángulo de los formalismos lógicos, como un conjunto de fórmulas que, junto a un conocimiento básico disponible, permiten derivar lógicamente el estado del mundo en cuestión (los síntomas por ej.), el problema que subsiste es hallar la "mejor" explicación en algun sentido (la mas simple, mas general, mas probable, etc.).

Reiter responde a esta observación dividiendo el problema de hallar la mejor teoría que soporte una evidencia dada en dos subproblemas :

a-Hallar y caracterizar el espacio de teorías posibles que soporten la evidencia.
b-Encontrar la "mejor" teoría en ese espacio.

Los sistemas de diagnóstico usuales, buscan converger a la "mejor" teoría sin explicitar el espacio de búsqueda (de hecho no realizan la discriminación de subproblemas). El rol de las lógicas no monotónicas reside en este caso en caracterizar dicho espacio. En [REI 86] se muestra que el espacio de teorías (explicaciones) posibles es precisamente el conjunto de extensiones de una formalización adecuada en lógica "default".

La segunda cuestión -hallar la "mejor" teoría- cae fuera del campo de la lógica no monotónica y requiere fuertes componentes heurísticas que definan "preferencias " o "precedencias " de teorías. En todo caso la delimitación explícita del espacio de búsqueda, puede ser un primer paso para encarar un estudio serio de todo el problema.

5. CONCLUSIONES.

De lo visto ,pareciera desprenderse que, si bien la no monotonicidad es mas la regla que la excepción en el razonamiento de sentido común, los intentos de fundamentar formalmente su aplicación son aun incipientes e imperfectos, como surge del sencillo caso de frame problem visto en la sección 4. Lo predominante es la implementación de mecanismos de inferencia no monotónica sin bases claras, entre los que se destaca la negación por falla de Prolog como el mas logrado. Esto es natural, ya que sería imposible y tampoco deseable, paralizar los desarrollos "prácticos" por falta de una teoría formal satisfactoria a cuyo surgimiento, por otra parte contribuyen dichos desarrollos.

McDermott en [MCD 86] expresa estas ideas para el caso de los sistemas de diagnóstico y el razonamiento abductivo como sigue:

"...Este estado de las cosas no evitará que sigamos escribiendo programas de diagnóstico médico. Pero si impedirá que los comprendamos. No existe teoría independiente alguna que podamos invocar para justificar las inferencias que hace el programa tácitas de abducción; estas teorías podrían ser las primeras teorías formales no triviales de abducción, si sólo pudiéramos hacerlas explícitas..."

6. BIBLIOGRAFIA.

[BOS 85]-Saturation, nonmonotonic reasoning and the closed world assumption. G.Bossu and P.Siegel. Artif.Intelligence 25(1985). *

[BRA 84]-A fundamental trade-off in KR and reasoning. R.Brachman y H.Levesque. 1984.In Readings in Knowledge Representation. *

[CLA 78]-Negation as failure.K.Clark.1978.(in Logic and

Data Bases eds. Gallaire y Minker). *

[HAN 86]-Default reasoning, nonmonotonic logics and the frame problem. S.Hanks and D.McDermott. 1986. (in Proc. Am. Assoc. for AI Nat. Conf.).

[HAY 79]-The Logic of frames.P.Hayes.1979.(In Frame Conceptions and Text Understanding. *

[KON 87]-On the relation between default theories and autoepistemic logics. 1987. SRI Int.,AI Center, Tech. Rep.

[KOW 83]-Amalgamating language and metalanguage.R.Kowalski.In Logic Programming (Eds.Clark and Tarnlund). *

[KOW 86a]-A logic-based calculus of events. R.Kowalsky and M.Sergot. New Generation Comp. 4(1986).

[KOW 86b]-Data Base updates in the event calculus. R.Kowalsky.1986. *

[LEV 84]-Foundations of a functionnal approach to KR. H.Levesque.Artif. Intelligence 23 (1984). *

[LEV 86]-Knowledge Representation and reasoning. H.Levesque.1986.(In Ann. Rev. Comput. Sci. 1:255-87. *

[LEV 86]-Making believers out of computers. H.Levesque.Artif. Intelligence (1986). *

[LEV 87]-All I know.Preliminary report. H.Levesque 1987. Univ. Toronto Tech. Rep.

[LIF 86]-On the satisfiability of circumscription. V.Lifschitz. Artif.Intelligence 28(1986).

[MCC 77]-Epistemological problems in AI.J.McCarthy. 1977.(In Proc. Int.It. Conf. Artif.Intell.-Cambridge-Mass.). *

[MCC 80]-Circumscription - a form of non monotonic reasoning J.McCarthy. Artif.Intelligence 13(1980).

[MCD 80]-Non monotonic logic I.D.McDermott y J.Doyle. Artif. Intelligence 13(1980).

[MCD 82]-Non monotonic logic II:non monotonic modal theories. D.McDermott. J.ACM 29(1). 1982.

[MCD 86]-A critique of pure reason. D.McDermott. 1986. Yale Univ. Tech.Rep.

[MIN 81]-A framework for representing Knowledge.M. Minsky. 1981.(In Mind Design).

[MOO 82]-The rol of Logic in Knowledge Representation and commonsense reasoning.R.Moore.1982. (In Proc.Nat. Conf.Am.Assoc.Artif. Intell.). *

[MOO 85]-Semantical considerations on no monotonic logic. R.Moore. Artif. Intelligence 25(1985). *

[NEW 81]-The knowledge level.A.Newell.The AI Magazine. Summer 1981. *

[NIL 80]-Principles of AI.N. Nilsson.Palo Alto.Calif. Tioga Eds.1980.

[REI 78a]-On reasoning by default.R.Reiter.1978. (In Proc.Conf. Theor. Issues Nat. Lang.Proc.- Urbana-Illinois). *

[REI 78b]-On closed world data bases.R.Reiter.1978. (in Logic and Data Bases eds.Gallaire y Minker). *

[REI 80]-A logic for default reasoning.R.Reiter. Artif.Intelligence 13(1980).

[REI 86]-A theory of diagnosis from first principles. R.Reiter. 1986. Artif.Intelligence (in press).

[REI 87]-Non monotonic reasoning.R.Reiter.1987.Unpublished. *

[SHO 86]-Reasoning about change: time and causation from the standpoint of Artificial Intelligence. Y.Shoham. 1986. Yale Univ.Ph.D. thesis.

[TUR 84]-Logics for Artificial Intelligence. R.Turner.1984.Ellis Horwood Ltd.London.GB. *

Instituto de Computación
Facultad de Ingenieria
Universidad de la República
Montevideo - URUGUAY

Msc. Juan Jose Cabezas
Palmas y ombues 5764
Montevideo Uruguay
Telf. 50 29 84

IGUALDAD DEFINICIONAL DE EXPRESIONES MATEMATICAS EN LA LOGICA INTUITIVA

RESUMEN

En este trabajo es presentada una especificación formal de expresiones matematicas e igualdad definicional, basada en la Teoria de Expresiones de Per Martin-Löf.

Per Martin-Löf ha desarrollado una teoria de expresiones donde las definiciones son eliminables y la igualdad entre expresiones es decidible. Dicha teoria se basa en el concepto de aridad (funcionalidad) de expresiones como mecanismo para controlar la formación de expresiones bien-formadas.

En esta monografía, se propone un tipo abstracto de expresiones, se analiza la expansión de expresiones (eliminación de definiciones), y finalmente se expone un procedimiento para decidir igualdad definicional entre expresiones de la misma aridad.

INTRODUCCION

La Teoria de Tipos de Per Martin-Löf puede ser interpretada como una aplicación de la Lógica Intuitiva a la ciencia de la Computación.

Esta teoria presenta un cuerpo unificado para implementar la actividad de especificación, construcción y verificación de programas. Fue originalmente desarrollada como una formalización de la matemática constructiva, con la cualidad que su lenguaje consagra la ocurrencia de demostraciones en las proposiciones, y como consecuencia, las proposiciones pueden expresar propiedades de las demostraciones. Esta facilidad tiene importantes consecuencias para la Teoria de Tipos considerada como un lenguaje de programación.

El lenguaje de la Teoria de Tipos

En la Teoria de Tipos, las expresiones son contruidas a partir de variables y constantes por medio de la aplicación y la abstracción funcional. Aunque el lenguaje, en apariencia, puede presentar semejanzas con el Cálculo Lambda, no es libre de tipos en el sentido de dicho cálculo. Posee más restricciones con respecto a las formas que una expresión bien-formada (wfe) puede tener.

El uso de la noción de "aridad de una expresión" -la cual describe su funcionalidad-, para imponer las restricciones apropiadas sobre las formas de aplicación que son legítimas, ha sido sugerido por Per Martin-Löf.

Dos importantes consecuencias de esta sugerencia concentran la atención de este trabajo:

- las definiciones en una expresión son eliminables
- la igualdad definicional entre expresiones es decidible.

1. EXPRESIONES E IGUALDAD DEFINICIONAL

Este capítulo está basado en Nordström [1].

Como son construidas las expresiones ?

Una manera posible y natural es permitir que las expresiones en general, sean construidas a partir de variables y constantes primitivas por medio de la aplicación y la abstracción funcional. Este es, también, el análisis que han hecho Church, Curry y sus seguidores en Lógica Combinatoria.

Sin embargo, hay dos consecuencias no naturales de esta definición. Una es que se puede aplicar, por ejemplo, la expresión succ, que representa la función sucesor, a tantas expresiones como se desee y formar expresiones como succ(x)(x), aún cuando la función sucesor solo tiene un argumento. La otra es que la autoaplicación está permitida; se puede formar, por ej., succ(succ). La autoaplicación, conjuntamente con la definición

$$((x)b)(a) :=: b[a/x]$$

donde :=: denota igualdad definicional, conduce a expresiones en las que no se pueden eliminar definiciones.

Esto puede constatarse a través del conocido ejemplo

$$((x)x(x))((x)x(x)) :=: ((x)x(x))((x)x(x)) :=: \ldots$$

Según Church [3] sabemos, por lo tanto, que si analizamos la igualdad definicional entre dos expresiones en esta teoria, no será siempre decidible si las expresiones son iguales definicionalmente.

Para resolver estas limitaciones Per Martin-Löf ha sugerido, retornando a Frege [4], que con cada expresión debería asociarse una aridad, describiendo la funcionalidad de la expresión.

1.1 La aridad de una expresión.

Utilizaremos las siguientes convenciones notacionales:

b(a1,..,an) en lugar de b(a1)(a2)... (an)
(x1,..,xn)d en lugar de (x1)(x2)... (xn)d
(w1,..,wn) en lugar de (w1)(w2)... (wn)

Si w1,..,wn (n>Ø) son aridades, entonces (w1,..,wn) es una aridad.
Para n=Ø se obtiene la aridad de una expresión saturada (completa) y se denota () o Ø.
Para n>Ø , (w1,..,wn) es la aridad de expresiones no saturadas

(incompletas, funcionales) las que aplicadas a n expresiones de aridad w1,..,wn , se saturan.
Cada variable y cada constante primitiva (predefinida) tienen una aridad asociada a ella.

A modo de ejemplo aclaratorio se presentan algunas expresiones y sus respectivas aridades.

expresión		aridad
x	-	Ø
1	-	Ø
succ	-	(Ø)
plus	-	(Ø,Ø)
succ(1)	-	Ø
plus(1)	-	(Ø)
plus(succ(1),x)	-	Ø

1.2 Que es una expresión de aridad w ?

Nordström [1] define lo que es una expresión de una cierta aridad; primero se refiere a expresiones saturadas, luego las no saturadas.

1. Si x es una variable de aridad (w1,..,wn) y a1,..,an son expresiones de aridadea w1,..,wn , respectivamente , entonces

x(a1,..,an) es una expresión saturada.

2. Si c es una constante de aridad (w1,..,wn) y a1,..,an son expresiones de aridades w1,..,wn . respectivamente , entonces

c(a1,..,an) es una expresión saturada.

3. Si, en una definición abreviatoria, el definiens (miembro derecho) es una expresión saturada, entonces también lo es el definiendum (miembro izquierdo).

4. Si b(x1,..,xn) es una expresión saturada, donde x1,..,xn son variables de aridades w1,..,wn , respectivamente, que no ocurren libres en b, entonces b es una expresión de aridad (w1,..,wn).

1.3 Reglas de formación de expresiones.

Nordtröm[1] define las reglas de formación de expresiones como sigue:

1. Variables - Una variable de aridad w es una expresión de aridad w.

2. Constantes Primitivas - Una constante primitiva de aridad w es una expresión de aridad w.

3. Constantes Definidas - Si b es una expresión saturada sin otras variables libres que x1,..,xn de aridades w1,..,wn respectivamente, y la constante c de aridad w1,..,wn es definida explicitamente por b, entonces c es una expresión de aridad w.

4. Abstracción - Si v es una expresión de aridad (v1,..,vn) y x1,..,xm son variables de aridad w1,..,wm respectivamente, entonces (x1,..,xm)b es una expresión de aridad (w1,..,wm,v1,..,vn)

5. Aplicación - Si c es una expresión de aridad (w1,..,wm,v1,..,vn) y x1,..,xm son variables de aridades w1,..,wm respectivamente, entonces c(x1,..,xm) es una expresión de aridad (v1,..,vn).

6. Substitución - Es posible substituir variables por expresiones: Si b es una expresión de aridad w, posiblemente conteniendo ocurrencias libres de las variables x1,..,xn de aridades v1,..,vm respectivamente, y a1,..,an son expresiones de estas aridades, entonces b[a1,..,am / x1,..,xm] es una expresión de aridad w.

---

2. UN TIPO ABSTRACTO DE EXPRESIONES

2.1 Sobre el diseño.

El primer paso en la especificación de expresiones es caracterizar un conjunto de funciones que definan la sintaxis abstracta de una expresión arbitraria.

Este conjunto de funciones sigen estrechamente las reglas de formación de expresiones descriptas en [1.3]. Sin embargo, no hay contraparte para la regla 6 -substitución- y la regla 5 se generaliza de forma tal que las expresiones son aceptadas como operandos.

2.1.1 Identificadores y listas · Se asume que los siguientes conjuntos y funciones están disponibles:

a. Id, el conjunto de los identificadores.

```
-- equal   :  Id x Id  --> Bool
```

b. Para cada conjunto A, List(A) es el conjunto de todas las listas con elementos de A. Se dispone, además, del siguiente grupo de funciones para construir y seleccionar listas.

Sea a,b:A l:List(A)

```
--   *       :      --> List(A)
--   .       :  A x  List(A)   --> List(A)

-- empty     :  List(A)  --> Bool
-- first     :  List(A)  --> A
-- rest      :  List(A)  --> List(A)

    empty(*)=true
    empty(a.l)=false

    first(a.l)=a

    rest(*)=*
    rest(a.l)=l
```

2.2 Constructores

De acuerdo con las reglas de formación de expresiones [1.3], podemos definir las siguientes funciones para construie expresiones a partir de variable y constantes.

Sea e:Exp, lx:List(Var), le:List(Exp).

```
-- cr_var        :  Id        -->    Var

-- cr_cont       :  Id        -->   Const

-- cr_abs        :  List(var)  x Exp  --> Exp

   cr_abs(lx,e) construye (lx)e

-- cr_appl       :  Exp  x  List(Exp)  -->  Exp

   cr_appl(e,le) construye e(le)
```

donde Var y Const son subconjuntos del conjunto Exp:

Exp - El conjunto de las expresiones.
Var - El subconjunto de las expresiones que son variables
Const - El subconjunto de las expresiones que son constantes

2.3 Selectores

Sea i:Id, e,a:Exp, le:List(Exp), lx:List(Var).

```
-- is_var        :   Exp   -->   Bool
   is_var(cr_var(i))=true
   para los demás casos se cumple is_var(a)=false

-- is_const      :   Exp   -->   Bool
   is_const(cr_const(i))=true
   para los demás casos se cumple is_const(a)=false

-- is_abs        :   Exp   -->   Bool
   is_abs(cr_abs(lx,e))=true
   para los demás casos se cumple is_abs(a)=false

-- is_appl       :   Exp   -->   Bool
   is_appl(cr_appl(e,le)=true
   para los demás casos se cumple is_appl(a)=false
```

Los selectores is_var, is_const, is_abs e is_appl cumplen con la propiedad de que dada una expresión arbitraria, uno y solo uno de ellos será verdadero:

(is var(e) XOR is const(e)) XOR (is abs(e) XOR is appl(e))=true

```
   indent        :   Exp  -->  Id
   indent(cr_var(i))=i
   indent(cr_const(i))=i
```

```
operator      :   Exp  --->  Exp
operator(cr_appl(e,le))=e

ops           :   Exp  --->  List(Exp)
ops(cr_appl(e,le))=le

expr          :   Exp  --->  Exp
expr(cr_abs(lx,e))=e

vars          :   Exp  --->  List(Var)
vars(cr_abs(lx,e))=lx
```

Se aclara que las funciones indent, operator, ops, expr y vars están indefinidas para ciertos argumentos.

---

3. EXPANSION DE EXPRESIONES

Nosotros hemos visto en [1.3] que si c es una constante definida de aridad (w1,..,wn), ella está definida explicitamente por una expresión saturada b sin otras ocurrencias de variables libres que no sean x1,..,xn de aridades w1,..,wn respectivamente.

Este capítulo está dirigido a demostrar que si e es una expresión saturada de la forma c(e1,..,en) y c es una constante definida, entonces c puede siempre ser eliminada de e. El proceso de eliminar una constante definida de una expresión es llamado expansión de expresiones.

3.1 Reglas de igualdad definicional

1. La igualdad definicional es una relación de equivalencia:

$$a :=: a \quad ; \quad \frac{a :=: b}{b :=: a} \quad ; \quad \frac{a :=: b \quad b :=: c}{a :=: c}$$

$$\frac{a :=: b}{(x)a :=: (x)b}$$

2. Si a y c son expresiones de aridad w y b una expresión de aridad (w) entonces

$$\frac{a :=: c}{b(a) :=: b(c)}$$

3. Beta-conversion

$$((x)b)(a) :=: b[a/x]$$

Las variables libres en a no pueden aparecer ligadas en b[a/x]

5.

$$(x)(b(x)) :=: b$$

x no debe ocurrir libre en b

6. Constantes definidas - Si c es una constante de aridad (w1,..,wn) y b es una expresión saturada (aridad=Ø) sin otras ocurrencias de variables libres que no sean x1,..,xn de aridades w1,..,wn respectivamente, entonces se puede plantear la definición explícita

c(x1,..,xn) :=: b

3.2 Expansión de una expresión

Supongamos que tenemos una expresión de la forma

(i) c(e1,..,en) · Ø

donde c es una constante definida de aridad (w1,..,wn), c(x1,..,xn) es un definiendum con definiens b, y e1,..,en son expresiones de aridades w1,..,wn, respectivamente. Con el soporte de [3.1] las siguientes iguadades son válidas

c(x1,..,xn) :=: b Reglas 2 y 6 [3.1]

(x1,..,xn)(c(x1,..,xn)) :=: (x1,..,xn)b Regla 5 [3.1]

(ii) c :=: (x1,..,xn)b

Ahora es posible eliminar c de (i) con ayuda de la igualdad (ii)

c(e1,..,en) :=: ((x1,..,xn)b)(e1,..,en) Regla 4 [3.1]

y por lo tanto

((x1,..,xn)b)(e1,..,en) :=: b[e1,..,en / x1,..,xn]

concluyendo

c(e1,..,en) :=: b[e1,..,en / x1,..,xn]

Se dice, entonces, que la expresión c(e1,..,en) ha sido expandida a una equivalente de la forma b[e1,..,en / x1,..,xn].

3.3 La lista de definiciones.

Cuando se ha de expandir una expresión debe ser posible el acceso a todas la constantes definidas y sus definiciones. Por esta razón, se supone disponible una lista de definiciones.

Se define el tipo de las definiciones, Def :

Sea i:Id, e:Exp, lx:List(Var).

```
--      cr_def      :  Id  x  List(Var)  x  Exp  ->  Def

---     def_id      :  Def  ->  Id
                    - def_id(cr_def(i,lx,e))= i

--      def_vars    :  Def  ->  List(var)
                    - def_vars(cr_def(i,lx,e))= lx

--      def_exp     :  Def  ->  Exp
                    - def_exp(cr_def(i,lx,e))= e
```

La lista de definiciones será accedida por las siguientes operaciones:

Sea c:Const, b:Exp, x1,..,xn:Var, d:Def, defs:List(Def).

```
is_def_const        :  Const  x  List(Def)  ->  Bool
                    -- is_def_const(c,defs) = true
                       si existe una definicion d en defs tal que
                       equal(ident(c),def_id(d)) = true


fetch_def           :  Const  x  List(Def)  ->  Def
                    - Si is_def_const(c,defs) = true
                       entonces fetch_def(c,defs) = d
                       y se cumple que
                       equal(ident(c),def_id(d)) = true
```

3.4 Eliminación de una constante definida en una expresión.

Si tenemos una expresión de la forma

c(e1,..,en)

donde c es una constante definida, c(x1,..,xn) es un definiendum con definiens b, y e1,..,en son expresiones, entonces se puede expandir dicha expresión de la manera expuesta en [3.2]:

c(e1,..,en) :=: b[e1,..,en / x1,..,xn]

Esta igualdad indica que la regla 6 de formación de expresiones -substitución- debe ser aplicada cuando se ha de expandir una expresión.

3.4.1 Substitución - Con la intención de definir una función que represente la regla 6 [1.3] capaz de evitar colisiones entre los nombres de variables libres y ligadas, se ha de seguir la estrategia

de avanzar paso a paso, comenzando por definir una función para un caso simple y restringido de substitución.

Paso I .- Si e es una expresión de aridad w, que posiblemente contiene ocurrencias libres de la variable x de aridad v, y es una variable de aridad v que no ocurre ligada en e, y si e', x', y' son representaciones de e, x e y, entonces simple subs(x',e',y') puede ser considerada como una representación de e[x/y].

simple_subs : Var x Exp x Var -> Exp

Paso II .- Si e es una expresión de aridad w, que posiblemente contiene ocurrencias libres de las variables x1,..,xn de aridad v1,..,vn, y1,..,yn son variables de estas aridades que no ocurren ligadas en e y ident(xi)<>ident(yk) para todo Ø<i<n+1, y si x1, e',y1 son representaciones de (x1,..,xn), e y (y1,..,yn), entonces vars_subs(x1,e',y1) puede ser considerada como una representación de e[y1,..,yn / x1,..,xn].

vars_subs : List(Var) x Exp x List(var) -> Exp

Paso III .- Si e es una expresión de aridad w, que posiblemente contiene ocurrencias libres de las variables x1,..,xn de aridades v1,..,vn, a1,..,an son expresiones de dichas aridades tales que ningún miembro de (x1,..,xn) ocurre libre en algún miembro de (a1,..,an), entonces exps_subs(x1,e',a1) puede ser considerada como una representación de e[a1,..,an / x1,..,xn].

exps_subs : List(Var) x Exp x List(Exp) -> Exp

Con el fin de prevenir colisiones entre los nombres de las variables, exps_subs puede substituir los nombres de variables ligadas por otros no utilizados previamente. Para crear estos nombres se supone disponible la operación new_names:

new_names : Integer -> List(Var)
- new_names(n) produce una lista de n variables cuyos nombres no han sido previamente utilizados.

Paso IV .- Finalmente, la función subst, que representa la regla ó [1.3], puede ser definida. Si e es una expresión de aridad w, que posiblemente contiene ocurrencias libres de las variables x1,..,xn de aridades v1,..,vn y a1,..,an son expresiones de estas aridades entonces

Sea e:Exp, x1,n1:List(Var), e1:List(Exp).

subst : List(var) x Exp x List(Exp) -> Exp

- subst(x1,e,e1) = aux_subst(x1,e,e1,new_names(length(x1)))

```
where
aux_subst(x1,e,e1,n1)=
      exps_subs(n1,vars_subs(x1,e,n1),e1)
```

3.4.2 Expand .- Con la función subst definida es posible presentar una definición de la función expand, que representa el proceso de expandir una expresión.

Sea c:Const una constante definida, e1:List(Exp), defs:List(Def).

```
    expand     :  Exp  x  List(Def)  -> Exp

  - expand(cr_appl(c,e1),defs) = subst( def_vars(fetch_def(c,defs)),
                                        def_exp(fetch_def(c,defs)),
                                        e1)
```

4. IGUALDAD DEFINICIONAL

4.1 Cuando son iguales dos expresiones de aridad w ?

Las siguientes reglas, según Nordström[1], hacen posible decidir cuando dos expresiones arbitrarias de la misma aridad son iguales. Utilizaremos la notación b - w por "b es una expresion de aridad w", y a :=: b - w por "a y b son expresiones iguales de aridad w".

1. Si x es una variable de aridad (w1,..,wn) y
   a1 :=: b1 - w1
   a2 :=: b2 - w2
   .. .. ..
   an :=: bn - wn
   entonces x(a1,..,an) :=: x(b1,..,bn) - Ø

2. Si x es una constante de aridad (w1,..,wn) y
   a1 :=: b1 - w1
   a2 :=: b2 - w2
   .. .. ..
   an :=: bn - wn
   entonces x(a1,..,an) :=: x(b1,..,bn) - Ø

3. Si a es un definiendum (miembro izquierdo) con definiens (miembro derecho) b, y b :=: c - Ø entonces a :=: c - Ø. Del mismo modo si a :=: b - Ø y b es el definiens de un definiendum c, entonces a :=: c - Ø.

4. Si b(x1,..,xn) :=: d(x1,..,xn) - Ø donde x1,..,xn son variables de aridades w1,..,wn, respectivamente, las cuales no ocurren libres en b o d entonces b :=: d - Ø.

Las reglas 1, 2 y 3 son definidas para expresiones saturadas (de aridad Ø). Como solamente variables, constantes primitivas y aplicaciones pueden ser expresiones saturadas, la regla 4 hace posible la decidibilidad de la igualdad definicional de expresiones no saturadas (de aridad (w1,..,wn)) aplicando las reglas 1, 2 y 3 de la siguiente manera:

Para decidir si las expresiones b y d de aridad (w1,..,wn) n>Ø son iguales, se aplican las reglas 1,2 y 3 a b(x1,..,xn) y d(x1,..,xn). Si b(x1,..,xn) :=: d(x1,..,xn) entonces b :=: d.

4.2 Forma reducida de expresiones.

Una expresión está en forma reducida si:

- es una variable o una constante primitiva
- es una aplicación a(e1,..,en) y el operador a es una variable o una constante primitiva
- es una abstracción (x1,..,xn)b y b no es una abstracción.

Las expresiones son preferidas en esta forma ya que, aplicando las reglas presentadas en [4.1], puede decidirse la igualdad definicional de dos expresiones, en cualquiera de los seis casos en que se presentan:

Caso 1.- Ambas expresiones son variables o contantes primitivas.

Caso 2.- Una expresión es una variable o una constante primitiva y la otra es una aplicación.

Caso 3.- Una expresión es una variable o una constante primitiva y lae otra es una abstracción.

Caso 4.- Ambas expresiones son aplicaciones.

Caso 5.- Una expresión es una aplicación y la otra es una abstracción.

Caso 6.- Ambas expresiones son abstracciones.

4.3 Un método para reducir una expresión.

Teorema : Existe un método para tranformar una expresión e de aridad w a una igual definicionalmente en forma reducida.

Definición I, Lema I y Lema II serán utilizados para obtener la forma reducida de expresiones:

Definicion I.

(a(e1,..,en))(en+1,..,em) :=: a(e1,..,en,en+1,..,em)

La función red_appl representa la definicion I:

Sea e,a:Exp, el1,el2:List(Exp).

```
red_appl     :     Exp    ->   Exp
             - Si e= cr_appl(cr_appl((a,el1),el2)
               entonces red_appl((e) = cr_appl(a,concat(el1,el2))
```

Lema I. - Regla 5 [3.1] y la Definición I sustentan el siguiente lema:

(x1,..,xn)((xn+1,..,xm)b) :=: (x1,..,xn,xn+1,..,xm)b

Prueba: las siguientes expresiones son definicionalmente iguales.

(x1,..,xn)((xn+1,..,xm)b)

Regla 5 [3.1]

(x1,..,xm)(((x1,..,xn)((xn+1,..,xm)b)) (x1,..,xm))

Definición I

(x1,..,xm)((((((x1,..,xn)((xn+1,..,xm)b))(x1,..,xn)))(xn+1,..,xm)))

Regla 5 [3.1]

(x1,..,xm)(((xn+1,..,xm)b)(xn+1,..,xm))

Regla 5 [3.1]

(x1,..,xm)b

La función red_abs representa el Lema I:

Sea e,a:Exp, xl1,xl2:List(Exp).

red_abs : Exp -> Exp
- Si e= cr_abs(xl1,cr abs(xl2,a))
  entonces red_abs((e) = cr_abs(concat(xl1,xl2),a)

Lema II.- Regla 4 [3.1] (Beta-conversion), Definición I y Lema I soportan el siguiente lema:

n=k, ((x1,..,xn)b)(e1,..,ek) :=: b[e1,..,ek/x1,..,xn]
n<k, ((x1,..,xn)b)(e1,..,ek) :=: (b[e1,..,en/x1,..,xn])(en+1,..,ek)
n>k, ((x1,..,xn)b)(e1,..,ek) :=: ((xk+1,..,xn)b)[e1,..,ek/x1,..,xk]

La función beta representa el Lema II:

Sea b,e:Exp, xl,xl1,xl2:List(Var), el,el1,el2:List(Exp).

beta : Exp -> Exp
- Si e = cr_appl(cr_abs(xl,b),el)
  entonces
  si length(xl)=length(el)
  entonces beta(e) = subst(xl,b,el)
  si length(xl) < length(el) y existen
    el1,el2 tales que el=concat(el1,el2) y
    length(xl) = length(el1)
  entonces beta(e) = cr_appl(subst(xl,b,el1),el2)
  si length(xl) > length(el) y existen
    xl1,xl2 tales que xl=concat(xl1,xl2) y
    length(xl1) = length(el)
   entonces beta(e) = (subst(xl1,cr_abs(xl2,b),el)

El siguiente procedimiento (Tabla 4.1) describe un método para

transformar una expresión e de aridad w en una definicionalmente igual en forma reducida.

Basado en este procedimiento, se define la función red_form que transforma una expresión en una equivalente en forma reducida, luego de un número finito de pasos.

Si e es	red_form(e,defs) =
1. - una variable o una constante primitiva.	e
Una aplicación cuyo operador puede ser:	
2.a- una variable o una constante primitiva.	e
2.b- una constante definida.	red_form(expand(e,defs),defs)
2.c- una aplicación.	red_form(red_appl(e),defs)
2.d- una abstracción.	red_form(beta(e),defs)
Una abstracción cuya expresión puede ser:	
3.a- una abstracción.	red_form(red_abs(e),defs)
3.b- una variable o una constante o una aplicación.	e

TABLA 4.1 - Una descripción de red_form.

4.4 Un algoritmo para igualdad definicional.

Sea e1 y e2 dos expresiones arbitrarias de la misma aridad. Para decidir si e1 y e2 son definicionalmente iguales, se transforman e1 y e2 a la forma reducida, y luego se aplican las reglas de igualdad [4.1] en los seis casos posibles que dos expresiones en forma reducida presentan:

Caso 1 - Si e1 y e2 son variables o constantes primitivas de aridad (w1,..,wn), entonces e1 y e2 son definicionalmente iguales si

$$e1(x1,..,xn) :=: e2(x1,..,xn) - \emptyset \qquad \text{Regla 4 [4.1]}$$

y esta igualdad se cumple si Regla 1 2 [4.1]

e1 y e2 son ambas variables y e1 = e2

o
e1 y e2 son ambas constantes primitivas y e1 = e2

donde x1,..,xn son variables de aridad w1,..,wn respectivamente, que no ocurren libres en e1 y e2.

Caso 2 - Si una de las expresiones es una aplicación b(a1,..,an) de aridad (w1,..,wk), y la otra es una variable o una constante primitiva d, entonces e1 y e2 son definicionalmente iguales si

e1(x1,..,xk) :=: e2(x1,..,xk) - Ø Regla 4 [4.1]

donde x1,..,xk son variables de aridad w1,..,wk respectivamente, que no ocurren libres en e1 y e2.

Pero
(b(a1,..,an))(x1,..,xk) :=: b(a1,..,an,x1,..,xk) Definición I
y
b(a1,..,an,x1,..,xk) y d(x1,..,xk) son definicionalmente iguales si y solo si n+k=k (n=Ø) y b y d son definicionalmente iguales (Reglas 1 y 2 [4.1]). Pero n=Ø no es válido ya que una aplicación tiene al menos un operando, n>Ø.
Conclusión: e1 y e2 no son definicionalmente iguales.

Caso 3 - Si una de las expresiones es una abstracción (x1,..,xn)b de aridad (w1,..,wk), y la otra es una variable o una constante primitiva de la misma aridad, entonces e1 y e2 son definicionalmente iguales si

e1(y1,..,yk) :=: e2(y1,..,yk) - Ø Regla 4 [4.1]
donde y1,..,yk son variables de aridad w1,..,wk respectivamente, que no ocurren libres en e1 y e2.

Pero
e1(y1,..,yk) :=: (e1(y1,..,yn))(yn+1,..,yk) Definición I
e1(y1,..,yk) :=: (e1(y1,..,yn))(yn+1,..,yk) Definición I
y si
e1(y1,..,yn) :=: e2(y1,..,yn) - (wn+1,..,wk) Regla 3 [3.1]
entonces
(e1(y1,..,yn))(yn+1,..,yk) :=: (e2(y1,..,yn))(yn+1,..,yk)
e1(y1,..,yk) :=: e2(y1,..,yk) - Ø
e1 :=: e2 (w1,..,wk)

Conclusión: e1 y e2 son definicionalmente iguales si e1(y1,..,yk) y e2(y1,..,yk) son definicionalmente iguales.

Caso 4 - Si e1:=:d(a1,..,am) y e2:=:b(g1,..,gn) son ambas aplicaciones de aridad (w1,..,wk), entonces e1 y e2 son definicionalmente iguales si m=n y si

e1(y1,..,yk) :=: e2(y1,..,yk) - Ø Regla 4 [4.1]
donde y1,..,yk son variables de aridad w1,..,wk respectivamente, que

no ocurren libres en e1 o e2.

Pero

```
        e1(y1,..,yk) :=: d(a1,..,an,y1,..,yk)      Definición I
        e2(y1,..,yk) :=: b(a1,..,an,y1,..,yk)      Definición I
```

y

```
        si        d :=: b                          Regla 1 o 2 [4.1]
                  a1 :=: g1
                  ..     ..
                  an :=: gn
                  y1 :=: y1
                  ..     ..
                  yk :=: yk
        entonces

  d(a1,..,an,y1,..,yk) :=: b(g1,..,gn,y1,..,yk)  - Ø  Regla 3 [4.1]
y
        e1(y1,..,yk) :=: e2(y1,..,yk)  - Ø
y
        e1 :=: e2  - (w1,..,wk)
```

Conclusión: e1 y e2 son definicionalmente iguales si m=n y si
a1 y b1 son definicionalmente iguales y
a2 y b2 son definicionalmente iguales y
.. ..
am y bn son definicionalmente iguales.

Caso 5 - Si una de las expresiones es una abstracción (x1,..,xn)d de aridad (w1,..,wk), y la otra es una aplicación, entonces e1 y e2 son definicionalmente iguales si e1(y1,..,yk) e2(y1,..,yk) son definicionalmente iguales (procedimiento similar al caso 3).
y1,..,yk son variables de aridad w1,..,wk respectivamente, que no ocurren libres en e1 o e2.

Caso 6 - si e1:=:(x1,..,xn)d y e2:=:(y1,..,yr)b son ambas abstracciones de aridad (w1,..,wk) entonces e1 y e2 son definicionalmente iguales si e1(z1,..,zm) y e2(z1,..,zm) lo son (procedimiento similar al caso 3).
k>=n,k>=r y m=max(n,r)
z1,..,zm son variables de aridad w1,..,wm respectivamente, que no ocurren libres en e1 o e2.

Observe que no es necesario conocer la aridad de las expresiones para decidir la igualdad definicional. Solamente se supone que las expresiones son de la misma aridad.

La función def equal.

La función def_equal, basada en este algoritmo, es definida para decidir si dos expresiones de la misma aridad son definicionalmente iguales:

Sea e1,e2:Exp, defs:List(Def)

```
def_equal   :   Exp  x  Exp  x  List(Def)  ->  Bool
            -  def_equal(red_form(e1), red_form(e2),defs)=true
                    si e1 y e2 son definicionalmente iguales
               def_equal(red_form(e1), red_form(e2),defs)=false
                    si e1 y e2 no son definicionalmente iguales
```

==========

REFERENCIAS

[1] B. Nordström, K. Petersson and J. Smith
An Introduction to Type Theory.
Programming Methodology Group. Chalmers University of Technology, Gothemburg-Sweden. (1983)

[2] B. Nordström and J. Smith
Propositions, Types and Specifications in Martin-Löf's Type Theory.
Dept. of Computer Sciences. Chalmers University of Technology, Gothemburg-Sweden. (1983)

[3] A. Church
An unsolvable problem of elementary number theory.
American Journal of Mathematics. Vol 58 pp. 345-363 (1936)

[4] G. Frege
From Frege to Goedel
A Source Book in Mathematical Logic 1879-1931
ed. J. van Heijenoort, Harvard University Press, Cambridge, Mass. (1967)

LA IGUALDAD EN TIPOS ABSTRACTOS DE DATOS

Rodrigo CARDOSO R.
Universidad de los Andes
Departamento de Sistemas
Apartado Aéreo 4976
Bogotá, D.E.
COLOMBIA

Resumen: Se presenta una definición de igualdad en tipos abstractos de datos (TADs) que soluciona dificultades prácticas en la utilización de la metodología de diseño e implementación que se expone en [Car86].

La nueva definición hace más fácil para el diseñador el decidir que el TAD construído refleja de manera adecuada el universo que pretende modelar. El poner en claro la definición de igualdad permite además que la prueba mecánica de teoremas sobre el TAD (v.gr.adecuación al modelo, verificación de implementaciones) se pueda realizar con buenos resultados prácticos.

Palabras claves: Tipos abstractos de datos, relaciones de igualdad.

****

0. Introducción.

En [Car86] se describe una metodología de diseño de tipos abstractos de datos (TADs) guiada por el conocimiento de un universo de objetos que se quiere modelar. La labor de diseño va de la mano de un proceso de ajuste entre la abstracción y la realidad, la metodología pretende explicitar criterios par juzgar la bondad de este ajuste.

Al llevar a la práctica aquellas ideas (construyendo herramientas para diseñar e implementar TADs) se descubre una dificultad asociada con la forma de decidir en qué caso dos términos de un TAD son iguales. En **3.** se expone un problema típico con la metodología anterior y se muestra como las reformas propuestas solucionan la dificultad.

En **1.** se da un resumen de lo necesario de [Car86] para entender este artículo. Las reformas anunciadas se incluyen en **2.**

1. Conceptos básicos.

$\langle Y,F \rangle$ es un **tipo abstracto de datos** (TAD) cuando Y es un conjunto (construíble) de símbolos y F un conjunto finito de operaciones sobre estos símbolos y tal vez otros TADs ya entendidos. El conjunto Y es el **tipo de interés** del TAD, y sobre él se cuenta con un orden bien fundado $.<_Y.$. Mientras no haya confusión Y denotará el TAD $\langle Y,F \rangle$. Cuando Y es un conjunto de objetos explícitos y F un conjunto de operaciones calculables sobre ellos se dice que Y es euna **estructura computacional** (EC).

Una de las reformas al la metodología de [Car86] es exigir además una relación de igualdad $.=_Y.$ definida sobre Y, no necesariamente la correspondiente al orden $.<_Y.$, que permite una revisión más efectiva de la adecuación del TAD al universo de objetos que se pretende modelar, a la vez que hace posible la demostración de teoremas del TAD que no son derivables con la sola aplicación de reglas de reescritura.

El conjunto de operaciones F es la unión de los conjuntos de operaciones **iniciales, constructoras, simplificadoras, y analizadoras** (I, C, S, A respectivamente). Es cómodo llamar $FG = I \cup C$ el conjunto de operaciones **generadoras** y $FS = S \cup A$ el conjunto de operaciones **selectoras**

El tipo de interés Y es el conjunto de términos sin variables generados por FG. Cada f∈FG se define por autodenotación (v.gr. para c∈C con funcionalidad c: Y x X ---> Y, el valor de c(y,x) es justamente la cadena de símbolos 'c(y,x)').

Las funciones en FS son definidas inductivamente sobre la complejidad de la construcción del tipo de interés. Las definiciones son dadas mediante un conjunto finito de axiomas $\mathcal{A}$. Cada axioma es de la forma

<lado derecho> = <lado izquierdo>

El <lado derecho> es un término cuyo símbolo de operación más externo es el de una selectora, y para cada selectora hay suficientes axiomas para garantizar una definición recursiva de la misma, con lo que se asegura la completitud suficiente del TAD. Obsérvese que no se dan axiomas para operaciones generadoras.

2. La igualdad en un TAD.

En [Car86] se definió una relación de "equivalencia semántica" de elementos de un TAD Y que presenta problemas al tratar de usarse para mostrar ciertos teoremas del TAD [Bar87]. A continuación se da una nueva definición igualmente plausible y factible de utilizar.

Como hipótesis adicional se requiere de cada TAD Xi conocido que sirva como parámetro a Y el tener definida una relación de igualdad decidible $=_{Xi}$.

Notación: Para $y = (y_1,...,y_{m-1}) \in Y^{m-1}$, $y_0 \in Y$, $1 \le j \le m$:

$$[y,y_0]_j = (y_1,...,y_{j-1},y_0,y_{j+1},...,y_m).$$

2.1 Definición

Sean t1, t2 ∈ F_* (términos sin variables),

y1, y2 ∈ Y,

f ∈ FS, con funcionalidad f: Y^m x $dom_X(f)$ ----> Z ($dom_X(f)$ es un producto cartesiano X1 x ... x Xr de TADs conocidos, Z = Y si f ∈ S, y Z = Xi si f ∈ A),

K un subconjunto de FS.

a) t1 →* t2 ssi t1 reduce a t2 por aplicación de axiomas

t1 =* t2 ssi Existe t tal que t1 →* t y t2 →* t.

b) $y1 =_f y2$ ssi Para todo $\mathbf{y} \in Y^{m-1}$, $x \in dom_X(f)$, $1 \le j \le m$:

$$f([\mathbf{y},y_1]_j,x) =^* f([\mathbf{y},y_2]_j,x), \text{ si } f \in S,$$
$$f([\mathbf{y},y_1]_j,x) =_{Xi} f([\mathbf{y},y_2]_j,x), \text{ si } f \in A.$$

c) $y1 =_K y2$ ssi Para todo $f \in K$: $y1 =_f y2$.

d) $y1 \equiv y2$ ssi $y1$ es **fuertemente equivalente** a $y2$
ssi $y1 =_{FS} y2$.

e) $y1 =_{K,f} y2$ ssi Para todo $\mathbf{y} \in Y^{m-1}$, $x \in dom_X(f)$, $1 \le j \le m$.

$$f([\mathbf{y},y_1]_j,x) =_K f([\mathbf{y},y_2]_j,x).$$

f) $.=_K.$ es una **igualdad** ssi

Para todo $f \in FS$: $y1 =_K y2 \Rightarrow y1 =_{K,f} y2$.

En este caso se dice que K es un **caracterizador**.

□

Con la anterior definición es concebible que en Y se puedan considerar varias relaciones de igualdad. Además de ser relación de equivalencia, una igualdad satisface una propiedad de sustitución entre términos relacionados, i.e. si dos términos son iguales, cualquier expresión que dependa de uno de ellos evalúa al mismo valor si se utiliza el otro en su lugar.

En [Car86] solo se aceptaba una igualdad, la aquí llamada equivalencia fuerte $.\equiv.$. Claramente vale para cualquier K, subconjunto de FS:

$$y1 \equiv y2 \Rightarrow y1 =_K y2,$$

y en particular para los caracterizadores. El concepto es aquí más general y exige del diseñador una última decisión para completar la definición del TAD Y: escoger un caracterizador que defina la igualdad $.=_Y.$.

La elección de un caracterizador no es tan difícil en la práctica, porque el diseñador debería tener claro qué propiedades son las que quiere observar para distinguir un objeto de otro en el universo que desea modelar. En últimas puede escoger siempre a FS como caracterizador, y la igualdad asociada es la equivalencia fuerte.

El ideal es elegir un caracterizador K tal que $y1 =_K y2$ si y solo si y1, y2 nombran un mismo objeto del mundo. Por supuesto esta es una propiedad apenas intuitivamente constatable, y en el mejor de los casos la demostración de una tal adecuación depende de una teoría para el mundo cuyo formalismo es mucho más general que el de los TADs, de modo que hay que usar conocimiento adicional para probar que ésto vale. A continuación se ilustra mediante un ejemplo cómo puede funcionar este método.

3. Un ejemplo.

El siguiente TAD modela los conjuntos finito de elementos de un TAD X.

Tipo Set [X]

* empty:		---->	Set
* insert:	Set x X	---->	Set
delete:	Set x X	---->	Set
has:	Set x X	---->	**bool**
union:	Set x Set	---->	Set

Axiomas

(1) delete(empty,x) = empty

(2) delete(insert(s,x1),x2) = **if** $x1 =_X x2$ **then** delete(s,x2)
else insert(delete(s,x2),x1)
fi

(3) has(empty,x) = **false**

(4) has(insert(s,x1),x2) = **if** $x1 =_X x2$ **then true**
else has(s,x2)
fi

(5) union(empty,s) = s

(6) union(insert(s1,x),s2) = insert(union(s1,s2),x)

finTipo

Para $a,b \in X$, el conjunto $\{a,b\}$ tiene dos nombres en Set[X]:

$$s1 = \text{insert}(\text{insert}(\text{empty},a),b)$$
$$s2 = \text{insert}(\text{insert}(\text{empty},b),a).$$

En [Car86] se sugirió que era posible demostrar que $s1 \equiv s2$ Los términos a cada lado de la equivalencia son irreducibles, porque no hay axiomas para generadoras. Si se intenta una demostración que chequee si s1 y s2 se portan en la misma forma para cada selectora, tal intento resulta fallido [Bar87].

Por ejemplo, debería suceder, para cualquier $c \in X$, que:

$$\text{delete}(s1,c) \equiv \text{delete}(s2,c).$$

Ahora bien.

$$\begin{aligned}\text{delete}(s1,c) &= \text{delete}(\text{insert}(\text{insert}(\text{empty},a),b), c)\\ &= \textbf{if } b =_X c \textbf{ then } \text{delete}(\text{insert}(\text{empty},a),c)\\ &\qquad \textbf{else } \text{insert}(\text{delete}(\text{insert}(\text{empty},a),c),b)\\ &\quad \textbf{fi}\end{aligned}$$

Y después de reescribir:

$$\begin{aligned}\text{delete}(s1,c) = \textbf{if } & b =_X c \wedge a =_X c \textbf{ then } \text{empty}\\ \textbf{elsf } & b =_X c \wedge a \neq_X c \textbf{ then } \text{insert}(\text{empty},a)\\ \textbf{elsf } & b \neq_X c \wedge a =_X c \textbf{ then } \text{insert}(\text{empty},b)\\ \textbf{elsf } & b \neq_X c \wedge a \neq_X c \textbf{ then } \text{insert}(\text{insert}(\text{empty},a),b)\\ \textbf{fi}\end{aligned}$$

Análogamente:

$$\begin{aligned}\text{delete}(s2,c) = \textbf{if } & b =_X c \wedge a =_X c \textbf{ then } \text{empty}\\ \textbf{elsf } & b =_X c \wedge a \neq_X c \textbf{ then } \text{insert}(\text{empty},a)\\ \textbf{elsf } & b \neq_X c \wedge a =_X c \textbf{ then } \text{insert}(\text{empty},b)\\ \textbf{elsf } & b \neq_X c \wedge a \neq_X c \textbf{ then } \text{insert}(\text{insert}(\text{empty},b),a)\\ \textbf{fi}\end{aligned}$$

Y para demostrar que las dos expresiones reducen a lo mismo en todos los casos se requiere que $s1 \equiv s2$ (obsérvese el caso $b \neq_X c \wedge a \neq_X c$).

Se mostrará ahora que K = {has} es un caracterizador para Set[X].

Hay que probar la siguiente proposición:

3.1 Proposición:

$s1 =_{has} s2 \quad \Rightarrow$ a) Para todo $x \in X$: $del(s1,x) =_{has} del(s2,x)$

b) Para todo $s \in Set$: $union(s1,s) =_{has} union(s2,s)$

y $union(s,s1) =_{has} union(s,s2)$.

La demostración puede ahora llevarse a cabo utilizando las definiciones y reglas de reescritura adecuadas. Hay otra manera de probar el resultado, apoyándose en el conocimiento que se puede tener del modelo pretendido, i.e. los conjuntos finitos de elementos de X. Se ilustrará cómo puede adelantarse una tal demostración.

Para el diseñador puede ser natural la siguiente definición de una función de representación, que establece qué conjunto se quiere representar con un nombre del TAD:

$$\Phi : Set[X] \dashrightarrow 2^X$$

definida recursivamente así:

$\Phi(empty) = \{\}$

$\Phi(insert(s,x)) = \Phi(s) \cup \{x\}$.

Y ahora es posible demostrar el siguiente resultado:

3.2 Lema:

a) $has(s,x)$ ssi $x \in \Phi(s)$.

b) $\Phi(delete(s,x)) = \Phi(s) \setminus \{x\}$

c) $\Phi(union(s1,s2)) = \Phi(s1) \cup \Phi(s2)$

d) $s1 =_{has} s2$ ssi $\Phi(s1) = \Phi(s2)$.

La prueba de **3.1** sigue fácilmente de este resultado

4. Conclusiones

La nueva definición de igualdad mejora los mecanismos de prueba de teoremas disponibles para un TAD. Para el diseñador es además más explícita la manera de constatar, al menos en forma intuitiva, en qué medida el TAD diseñado abstrae el universo de objetos que se pretende modelar.

El hecho de evitar proporcionar axiomas para operaciones generadoras permite hacer inducción sobre el tipo de interés, tanto para definir el significado de las operaciones selectoras como para llevar a cabo demostraciones de teoremas sobre el TAD.

Cuando el diseñador acepta que un caracterizador K define un igualdad satisfactoria en un TAD Y, está al mismo tiempo aceptando que el universo observable es isomorfo a $Y/=_K$. También se está entendiendo que el universo es un modelo inicial del TAD Y, en el sentido de que dos objetos son iguales en el universo si se puede demostrar la igualdad de los nombres correspondientes en el TAD, y esta igualdad se define como ausencia de diferencias en las características que el observador considera relevantes.

Debe ser claro, además, que entre más pequeño sea un conjunto caracterizador la igualdad asociada es más sencilla de utilizar. Por ésto es recomendable buscar caracterizadores minimales. Así, si K es un caracterizador y $f \in K$, si $.=_K.$ discrimina tanto como $.=_{K,f}.$, entonces es preferible elegir a $K \setminus \{f\}$ como caracterizador.

BIBLIOGRAFIA

[Bar87] Barón, R., "Una herramienta para el diseño de tipos abstractos de datos", Tesis de grado, Universidad de los Andes, Bogotá, 1987.

[Bau82] Bauer, F.L., Wössner H., Algorithmic Language and Program Development, Springer Verlag, 1982.

[Car86] Cardoso, R., "Diseño e implementación de tipos abstractos de datos", CLEI 1986, Montevideo

****

PROCEDIMIENTOS DE DECISION PARA LA EQUIVALENCIA Y LA CONGRUENCIA OBSERVACIONAL DE TERMINOS REGULARES DE CCS

A. SORIANO

UNIVERSIDAD CENTRAL DE VENEZUELA

RESUMEN

Se presenta la descripción de procedimientos de decisión para la equivalencia y la congruencia observacional de CCS. Dichos procedimientos de decisión se basan en transformaciones realizadas sobre los sistemas de transiciones que representan los comportamientos descritos por los términos, y en la noción de bisimulación. Estas transformaciones son de dos tipos: unas correspondientes a la eliminación de bucles y cadenas de acciones no visibles y la otras correspondientes a la eliminacion de duplicidades de un mismo comportamiento.

INTRODUCCION.

Actualmente con el desarrollo de los sistemas distribuídos y la concepción de lenguajes de programación capaces de expresar el paraleleismo, es posible construir sistemas cada vez más poderosos y rápidos. Sin embargo, estos sistemas son cada vez mas complejos, de allí que hoy por hoy en la concepción de sistemas comunicantes, es indispensable de realizar una fase de verificación que permita garantizar la fiabilidad de los sistemas.

La verificación de sistemas comunicantes es hoy en día un campo activo de investigación. En el presente trabajo nos interesamos por el problema de la verificación automática de sistemas comunicantes regulares.

Verificar la descripción de un sistema consiste en compararla con una espécificación. Desde un punto de vista teórico, es posible distinguir entre dos tratamientos diferentes. Un tratamiento basado en la lógica [AS 79] [Pn 77] [Ab 79] donde las propiedades a verificar se representan por un conjunto de aserciones sobre el comportamiento. Este tratamiento permite verificar propiedades generales de los sistemas como por ejemplo la ausencia de abrazo mortal. El otro tratamiento se basa en teorías algebráicas [Mi 80] [Mi 83] [BK 85] [AB 84] [BH 84] en las que las descripciones de los sistemas y sus espécificaciones se escriben como términos de un álgebra y las pruebas se realizan en base a una noción de equivalencia entre los términos del álgebra; este enfoque permite verificar propiedades específicas de los sistemas, como por ejemplo, la espécificación del servicio de un protocolo de comunicación. En lo sucesivo, consideraremos esta segunda forma de especificación.

En la literatura, hay varios ejemplos de verificación de sistemas comunicantes basados en un álgebra [Mi 80] [BK 84] [Ko 85]. Pero el problema principal en este tipo de pruebas está ligado a la manipulación de términos, y cuando más complejos son los sistemas, más aumenta cantidad de cálculo necesaria para la verificación; de ahí la necesidad de su automatización.

Este trabajo se orienta en esta dirección y su objetivo es el describir

un procedimiento de decisón para la equivalencia y la congruencia observacional de términos regulares del cálculo CCS de Milner [Mi 80].

CCS [Mi 80] es un cálculo para la descripción, la espécificación y la verificación de sistemas comunicantes asincronos. Este cálculo está basado en dos ideas principales: la observación y la comunicación sincrónica.

Informalmente, en CCS, puede describirse el comportamiento observable de un sistema (noción de observación). Es a partir de esta noción de observación que se definen los criterios de comparación de sistemas (equivalencia y congruencia observacional).

La descripción de un sistema puede hacerse también por composición paralela (operación que se escribe como ||) de procesosque se comunican entre ellos. La comunicación de prosesos se considera como una acción indivisible del sistema (idea de comunicación sincronizada), esta acción se denomina comunicación interna y se denota por "τ".

De una manera mas formal, las verificaciones de sistemas comunicantes consisten en transformar y comparar un término que describe un sistema con su especificación representada por otro término.

Recientemente, se han publicado trabajos [HM 85] [BK 85] [Mi 85] en los que se presenta una axiomatización completa para la equivalencia y la congruencia observacional, sin embargo, es importante disponer de procedimientos de verificación automatizables.

Este trabajo se orienta en este sentido y en él se describen procedimientos de decisión para la equivalencia y la congruencia observacional de términos regulares de CCS. Los procedimientos que se describen se basan en la aplicación de transformaciones ad-hoc definidas sobre *relaciones de transición* representando los comportamientos de los términos CCS y en la noción de *bisimulación*.

El trabajo consiste de tres partes: en la primera parte se realiza una descripción del algebra de CCS y la definición y axiomatización de la equivalencia y la congruencia observacional, en la segunda parte se describen los procedimientos de decisión de la equivalencia y la congruencia observacional y por último se da una conclusión.

I.- CCS: "Calcul of Communicating Systems".

CCS es un álgebra de términos provista de una relación de congruencia. La semántica operacional de su lenguaje le permite interpretar los términos como procesos (sistemas de transición comunicantes). La relación de congruencia permite la comparación de los términos.

En este trabajo, nos restringimos al álgebra de procesos que representa los comportamientos regulares.

I.1.- Algebra para CCS.

Sea:

- D, un conjunto de nombres de *acciones* o *puertos de comunicación* cuyos elementos se identifican por letras minúsculas,
- ~D, un conjunto de nombres de puertos llamados *complementarios* cuyos

elementos son los elementos de D precedidos por el símbolo "~",

- una biyeccion:

α <---> ~α, donde $\alpha \in D$,

~$\alpha \in$ ~D,

se dice que"~α" es el *puerto complementario* de "α",

ésto nos permite definir la interacción entre procesos,

- τ, ($\tau \notin D$ et $\tau \notin$ ~D) una etiqueta que representa una *acción no observable* resultante de una comunicación interna entre α y ~α, $\alpha \in D$,
- P, el conjunto de etiquetas de puertos de comunicación, $P = D \cup$ ~D,
- P_τ $P_\tau = P \cup \{\tau\}$,
- F, el conjunto de funciones de redefinición f tales que:

  f: $L \subseteq P$ ----> $M \subseteq P$

  f es una biyección,

  f respeta los complementos definidos por la biyección,
- V, el conjunto de variables de comportamiento,
- las firmas, $\Sigma_1 = (P_\tau, nil, +)$

  $\Sigma_2 = (P_\tau, nil, +, ||, [\,], \backslash)$

donde:

nil es una constante,

+ et || son operadores binarios,

$\forall \alpha \in P_\tau$ α, $\forall f \in F$ [f], $\forall \alpha \in P$ $\backslash\alpha$, son operadores unarios.

Los operadores + y || son operadores infijos, α es un operador prefijo, [f] y $\backslash\alpha$ son operadores postfijos. El símbolo "." se utiliza como separador después del operador α, por ejemplo se escribe α.nil y no αnil.

La prioridad es creciente para los opeadores '+' y '||', '[f]' y '$\backslash\alpha$', 'α.'. Los operadores '+', '||', '[f]' y '$\backslash\alpha$' son asociativos a izquierda y el operador 'α.' es asociativo a derecha.

Representamos por:

T_f, el álgebra de firma Σ_2. Los elementos de T_f se llaman *términos finitos*.

Se dice que un término finito está escrito en *forma de suma* si:

- sea t=nil,
- sea $t = \Sigma_{i \in I} \alpha_i.t_i$ y cada t_i es un término finito escritos en forma de suma.

T_S, es el subconjunto de términos finitos en forma de suma. ($T_S \subseteq T_f$).

$T[\Sigma_1,V]$, el álbegra de firma $\Sigma_1 \cup V$. ($T_S \subseteq T[\Sigma_1,V]$).

$T[\Sigma_2,V]$, el álgebra de firma $\Sigma_2 \cup V$. ($T[\Sigma_1,V] \subseteq T[\Sigma_2,V]$ y $T_f \subseteq T[\Sigma_2,V]$).

Los elementos de $T[\Sigma_1,V]$ y los de $T[\Sigma_2,V]$ son expresiones a

partir de las cuales se construyen los términos que definen comportamientos infinitos.

T'_r, el conjunto de pares de la forma: $u = (t,E)$

donde $t \in T[\Sigma_2,Y]$,

E es un conjunto de ecuaciones de la forma: $\{y_i=t_i\}_{i=1,...,n}$

donde $y_i \in Y$ y $t_i \in T[\Sigma_1,Y]$ y $\forall i,j \; i \neq j \Rightarrow y_i \neq y_j$. Se dice que t_i es la *definición* de y_i.

Los elementos de T'_r definen los comportamientos infinitos. Los pares $(t,\emptyset) \in T'_r$ (*términos finitos*) sont escritos simplemente t. ($T_f \subseteq T'_r$).

Para los términos de T'_r se introducen las nociones de *variable útil, variable inutil, variable libre, término cerrado, variable no guardada, término guardado y término bien guardado*. Estas nociones se utilizan para definir los términos regulares.

Para $t \in T[\Sigma_2,Y]$, sea var(t) el conjunto de variables de t.

Para un término u=(t,E), se dice que una variable y es *útil* si y solamente si $y \in \bigcup_{j=0}^{card(E)} \rho^j(var(t))$, donde $\rho(X)$ se define como:

$\rho(X)=\{z \mid y_i \in X \text{ y } y_i=t_i \text{ y } z \in var(t_i)\}$,

ρ^0 es la identidad

$\rho^{k+1}(X) = \rho(\rho^k(X))$, para k>1.

Se dice que una variable y est *inutil* en un término u=(t,E) si y solamente si $\exists t_i$ tal que $y=t_i$ pertenece a E y y no es una variable útil del término u.

Se dice que una variable y es *libre* en un término $(t,E) \in T'_r$ si y solamente sino existe j tal que $y=t_j \in E$.

Se dice que un término es *cerrado* si y solamente si no contiene variables libres.

Se dice que una variable y *no es guardada* en un término u=(t,E) si y solamente si y aparece en t o en la definición de una variable útil de u sin estar precedida por una acción.

Se dice que un término u=(t,E) es *guardado* si y solamente si toda variable no guardada del término puede reemplazarse por nil o por una expresion de la forma $\Sigma \alpha_i.t_i$, oar substitución repetida de variables por sus definiciones.

Se dice que un término u=(t,E) es *bien guardado* si y solamente si él es guardado y toda variable guardada por τ en él, puede ser reemplazada por nil o por una expresión de la forma $\Sigma_{i=1,...,n} \; \alpha_{i1}. \ldots . \alpha_{ik}.t_i, \alpha_{ik} \neq \tau$.

Sobre los términos de T'_r definimos los operadores α, +, ||, \, [f] de la manera siguiente:

$\alpha(t,E) = (\alpha t, E)$

$(t_1,E_1) + (t_2,E_2) = (t_1+t_2,\ E_1 \cup E_2)$, si $\nexists x_i \in Y$ tal que $x_i=t_i \in E_1$ y $x_i=t'_i \in E_2$

$(t_1,E_1) \parallel (t_2,E_2) = (t_1 \parallel t_2,\ E_1 \cup E_2)$, si $\nexists x_i \in Y$ tal que $x_i=t_i \in E_1$ y $x_i=t'_i \in E_2$

$(t,E)\backslash\alpha = (t\backslash\alpha, E)$

$(t,E)[f] = (t[f], E)$.

- T_r, es el subconjunto de términos cerrados y guardados de T'_r.
  Los elementos de T_r son los términos regulares de CCS.

I.2.- Semántica operacional de los operadores.

La semántica operacional [Pl 81] de CCS se da a través de relaciones de transición $\xrightarrow{\alpha}$, $\alpha \in P_\tau$. Estas relaciones son las más pequeñas relaciones binarias sobre T_r definidas por las reglas:

$$\frac{t \xrightarrow{\alpha} t'}{(t,E) \xrightarrow{\alpha} (t',E)}$$

$$\frac{\forall x=t' \in E \text{ et } t[t'/x] \xrightarrow{\alpha} t''}{(t,E) \xrightarrow{\alpha} (t'',E)}$$

(α): $\alpha.t \xrightarrow{\alpha} t$ (ACCION)

(+):

(1) $$\frac{t_1 \xrightarrow{\alpha} t'_1}{t_1+t_2 \xrightarrow{\alpha} t'_1}$$

(2) $$\frac{t_2 \xrightarrow{\alpha} t'_2}{t_1+t_2 \xrightarrow{\alpha} t'_2}$$

(ESCOGENCIA NO DETERMINISTA)

(II) (1) $$\frac{t_1 \xrightarrow{\alpha} t'_1}{t_1 \| t_2 \xrightarrow{\alpha} t'_1 \| t_2}$$

(2) $$\frac{t_2 \xrightarrow{\alpha} t'_2}{t_1 \| t_2 \xrightarrow{\alpha} t_1 \| t'_2}$$ (COMPOSICION PARALELA)

(3) $$\frac{t_1 \xrightarrow{\alpha} t'_1 , t_2 \xrightarrow{\sim\alpha} t'_2}{t_1 \| t_2 \xrightarrow{\tau} t'_1 \| t'_2}$$

(I) $$\frac{t \xrightarrow{\alpha} t' , \alpha \neq \beta}{t \backslash \beta \xrightarrow{\alpha} t' \backslash \beta}$$ (BORRADO)

(I) $$\frac{t \xrightarrow{\alpha} t'}{t[f] \xrightarrow{f(\alpha)} t'[f]}$$ (REDEFINICION)

Notación:

- Sea $s = \alpha_1 \ldots \alpha_n$, $n>0$ y $\alpha_i \in P$
  entonces $p \xrightarrow{\tau}{}^{*} \xrightarrow{\alpha_1} \xrightarrow{\tau}{}^{*} \cdots \xrightarrow{\alpha_n} \xrightarrow{\tau}{}^{*} p'$
  se escribe como $p \Rightarrow^{s} p'$.
- $t \xrightarrow{\tau}{}^{*} t'$ se escribe como $t \Rightarrow t'$.
- $t \xrightarrow{\tau}{}^{+} t'$ se escribe como $t \Rightarrow^{\tau} t'$.

I.3.- Relaciones de equivalencia y de congruencia.

R. Milner define relaciones de equivalencia y de congruencia sobre T'_r. Estas relaciones definen criterios de comparación entre términos. A continuación se presentan tres relaciones definidas para T'_r. Estas relaciones son:

- la *congruencia fuerte*
- la *equivalencia observacional*
- la *congruencia observacional*

I.3.1. *Congruencia Fuerte.*

Sean: $p, q \in T'_r$,

R una relación binaria sobre T'_r y $F(R)$ la función:

$(p,q) \in F(R)$ ssi (i) $p \xrightarrow{\alpha} p'$ implica $\exists q'\ (q \xrightarrow{\alpha} q'$ y $(p',q') \in R)$ y

(ii) $q \xrightarrow{\alpha} q'$ implica $\exists p'\ (p \xrightarrow{\alpha} p'$ y $(p',q') \in R)$,

Definición I.1.

Se dice que R es una *bisimulación* si y solamente si $R \subseteq F(R)$

Definición I.2.

La *equivalencia fuerte* (escrita ~) se define como la más grande relación de bisimulación, es decir:

$$\sim = \cup \{ R \subseteq T'_r \times T'_r \mid R \subseteq F(R) \}.$$

La relación ~ existe y es única porque F es monótona en el reticulado de las relaciones binarias con la inclusión de conjuntos ($\subseteq$).

Para los términos pertenecientes a T_f, la axiomatización de la *equivalencia fuerte* esta dada por: (A_1)-(A_{16}) de la Tabla 1. Esta relación de equivalencia es una relación de congruencia para los operadores α, + et ||. En lo sucesivo se denominará *congruencia fuerte*.

I.3.2.- *Equivalencia Observacional.*

La equivalencia observacional se define exactamente como la congruencia fuerte, excepto que se utilizan las relaciones $=\alpha\Rightarrow$ et $\Rightarrow$ en lugar de $\xrightarrow{\alpha}$ et $\xrightarrow{\tau}$. La equivalencia observacional se escribe como $\approx$.

Proposición I.1.

Para dos términos $t_1, t_2 \in T_r$, $t_1 \approx t_2$ si y solamente si se verifican las condiciones siguientes:

(i) $t_1 =\alpha\Rightarrow t'_1$ implica $(\alpha=\tau$ y $t'_1 \approx t_2)$

o $\exists t'_2 (t_2 =\alpha\Rightarrow t'_2$ y $t'_1 \approx t'_2)$

(ii) $t_2 =\alpha\Rightarrow t'_2$ implica $(\alpha=\tau$ y $t'_2 \approx t_1)$

o $\exists t'_1 (t_1 =\alpha\Rightarrow t'_1$ y $t'_1 \approx t'_2)$.

♦

Proposición I.2.

$p \sim q$ implica $p \approx q$. ♦

La relación de equivalencia observacional no es una realción de congruencia. Por ejemplo: $a.nil \approx \tau.a.nil$ pero, $b.nil + a.nil \not\approx$

observacionalmente equivalente a b.nil + τ.a.nil.

Para términos pertenecientes a T_f, La equivalencia observacional esta axiomatizada por: (A_1)-(A_{20}) de la Tabla 1.

I.3.3.- *Congruencia Observacional.*

La *congruencia observacional* (escrita $\approx^c$) se define como la mayor relación de congruencia para los operadores α, + et || incluida en la equivalencia observacional.

Proposition I.3.

Para dos términos $t_1, t_2 \in T_r$, $t_1 \approx^c t_2$ si y solamente si $\forall \alpha \in P_\tau$ se verifican la condiciones siguientes:

(1) $t_1 \xrightarrow{\alpha} t'_1$ implica $\exists t'_2 (t_2 = \alpha \Rightarrow t'_2$ y $t'_1 \approx t'_2)$

(2) $t_2 \xrightarrow{\alpha} t'_2$ implica $\exists t'_1 (t_1 = \alpha \Rightarrow t'_1$ y $t'_1 \approx t'_2)$

◆

En la proposición siguiente, damos una nueva caracterización para la congruencia observacional. Esta caracterización se utilizará en la definición del procedimiento de decisión de la congruencia observacional.

Proposition I.4.

Para dos términos $t_1, t_2 \in T_r$, $t_1 \approx^c t_2$ si y solamente si se verifican las condiciones siguientes:

- $t_1 \approx t_2$,
- $t_1 \xrightarrow{\tau} t'_1$ implica $\exists t'_2$ tal que $t_2 = \tau \Rightarrow t'_2$ y $t'_1 \approx t'_2$,
- $t_2 \xrightarrow{\tau} t'_2$ implica $\exists t'_1$ tal que $t_1 = \tau \Rightarrow t'_1$ y $t'_1 \approx t'_2$.

◆

Para los términos pertenecientes a T_f, la axiomatización de la equivalencia observacional está dada por: (A_1)-(A_{19}) de la Tabla 1.

Milner a demostrado la completitud de una axiomatización para la congruencia observacional para los términos recursivos. Esta axiomatización está definida para términos recursivos escritos de manera diferente a la nuestra, sin embargo, los axiomas y reglas de inferencia de nuestro formalismo pueden obtenerse de los suyos por transformaciones sintácticas. De esta manera, una axiomatización completa para la congruencia observacional de términos de T_r está dada por los axiomas (A_1)-(A_{19}) de la Tabla 1 y los axiomas y reglas de inferencia dados en la Tabla 2.

Cabe destacar que los axiomas (A_6), (A_7), (A_{12}), (A_{14}), (A_{16}) indicando las propiedades de asociatividad, commutatividad y distributividad de los operadores ||, \ y [ ] no pueden deducirse de los otros axiomas; pero en lo sucesivo no se utilizarán dichos axiomas porque todas sus instancias pueden deducirse.

TABLA 1:

(+):	(A_1):	$t_1 + (t_2 + t_3) = (t_1 + t_2) + t_3$
	(A_2):	$t_1 + t_2 = t_2 + t_1$
	(A_3):	$t + t = t$
	(A_4):	$t + nil = t$
(||):	(A_5):	Si t_1 y t_2 son de la forma: $t_1 = \sum_{i=1,\ldots,n} \alpha_i . t'_i$ et $t_2 = \sum_{j=1,\ldots,n} \beta_j . t'_j$ entonces: $t_1 \| t_2 = \sum \alpha_i . (t'_i \| t_2) + \sum \beta_j . (t_1 \| t'_j) + \sum \tau . (t'_i \| t'_j)$ $\alpha_i = \sim\beta_i \wedge \sim\alpha_i = \beta_i \wedge i=1,\ldots,n \wedge j=1,\ldots,n$
	(A_6):	$t_1 \| t_2 = t_2 \| t_1$
	(A_7):	$t_1 \| (t_2 \| t_3) = (t_1 \| t_2) \| t_3$
	(A_8):	$t \| nil = t$
([]):	(A_9):	$(\alpha.t)[f] = f(\alpha).(t[f])$
	(A_{10}):	$(t_1 + t_2)[f] = t_1[f] + t_2[f]$
	(A_{11}):	$nil[f] = nil$
	(A_{12}):	$(t_1 \| t_2)[f] = t_1[f] \| t_2[f]$
(\):	(A_{13}):	$(\alpha.t)\backslash\beta = \begin{cases} nil & \text{si } \alpha = \beta \text{ o } \alpha = \sim\beta \\ \alpha.(t\backslash\beta) & \text{si no} \end{cases}$
	(A_{14}):	$(t_1 + t_2)\backslash\alpha = t_1\backslash\alpha + t_2\backslash\alpha$
	(A_{15}):	$nil\backslash\alpha = nil$
	(A_{16}):	$t\backslash\alpha_1\backslash\alpha_2 = t\backslash\alpha_2\backslash\alpha_1$
(τ):	(A_{17}):	$t + \tau.t = \tau.t$
	(A_{18}):	$\alpha.(t_1 + \tau.t_2) = \alpha.(t_1 + \tau.t_2) + \alpha.t_2$
	(A_{19}):	$\alpha.\tau.t = \alpha.t$
	(A_{20}):	$\tau.t = t$

TABLA 2:

$$(1): \quad \frac{t_1 = t_2}{(t, E \cup \{x_1 = t_1\}) = (t, E \cup \{x_2 = t_2\})}$$

$$(2): \quad (t, E \cup \{x_1 = t_1, x_2 = t_2\}) = (t, E \cup \{x_1 = t_1, x_2 = t_2[^{t_1}/_{x_1}]\})$$

$$(3): \quad \frac{(z_i, E) = (t_i[^{z_j}/_{x_j}]_{j=1,\ldots,n}, E)}{(z_i, E) = (x_i, \{x_j = t_j\}_{i=1,\ldots,n})} \quad var(t_i) \{x_1, \ldots, x_n\}$$

$$(4): \quad \frac{x \notin \text{útil}(t, E)}{(t, E) = (t, E \cup \{x = t_1\})}$$

$$(5): \quad (t, E \cup \{x = \tau.x + t_1\}) = (t, E \cup \{x = \tau.t_1\})$$

$$(6): \quad (t, E \cup \{x = \tau.(x + t_1) + t_2\}) = (t, E \cup \{x = \tau.x + t_1 + t_2\})$$

II. Verificación de términos regulares

En esta parte, se presenta un procedimiento de decisión para la congruencia fuerte, la equivalencia observacional y la congruencia observacional de términos que describen comportamientos regulares (*términos regulares*).

Este procedimiento utiliza la noción de *sistema de transiciones* para representar las derivaciones de los términos y la noción de *bisimulación* para compararlos. A partir de estas nociones y basándose en la propiedad I.1., se obtiene directamente un procedimiento de decisión para la congruencia fuerte. Considerando las proposiciones I.2. y I.4. definimos las *transformaciones EO* y *CO* que permiten decidir respectivamente la equivalencia y la congruencia observacional por *simulación*.

Para decidir cada una de las equivalencias que se han definido sobre T_r, procedemos en tres etapas:

1. Construcción del *sistema de transiciones* asociado a cada término a comparar.
2. Aplicación de transformaciones ad hoc sobre los sistemas de

transición construídos.
3. Decidir la equivalencia correspondiente por bisimulación entre los sistemas de transiciones transformados.

II.1.- Construcción de sistemas de transiciones.

Definición II.1.

Un *sistema de transiciones* S_u *asociado a un término* $u=(t,E)\in T_r$ se define de la manera siguiente:

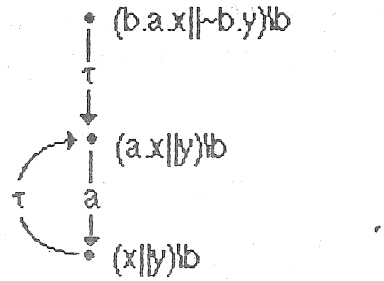
$$S_u = (Q, \cup_{\alpha \in P_\tau} R_\alpha, t)$$

donde:

- Q es un conjunto de estados correspondiente al conjunto de términos obtenidos por derivación a partir de u,
- $R_\alpha \subseteq Q \times Q$ se define como: $(q',q'') \in R_\alpha$ si y solamente si $(t',E) \xrightarrow{\alpha} (t'',E)$ y $q'=(t',E)$ y $q''=(t'',E)$.

  Para indicar $(p,q)\in R_\alpha$ se utiliza también $p \xrightarrow{\alpha} q$ y de maniera análoga se utilizan las notaciones $=\alpha\Rightarrow$ et $\Rightarrow$.
- u es el estado inicial del sistema.

Por ejemplo, el sistema de transiciones asociado al término: $((b.a.x||\sim b.y)\backslash b,\{x=b.a.x, y=\sim b.y\})$, es:



Es posible extender las relaciones $\sim$, $\approx$ et $\approx^c$ definidas sobre los términos, a los sistemas de transiciones correspondientes; es decir, para los términos $t_1, t_2 \in T_r$,

(i) $t_1 \sim t_2$ si y solamente si $S_{t1} \sim S_{t2}$

(ii) $t_1 \approx t_2$ si y solamente si $S_{t1} \approx S_{t2}$

(iii) $t_1 \approx^c t_2$ si y solamente si $S_{t1} \approx^c S_{t2}$

II.2.- Transformaciones de sistemas de transiciones.

En esta parte, se definen las transformaciones *EQ* y *CQ*. Se dan algunas propiedades interesantes de dichas transformaciones y su utilización para decidir la equivalencia y la congruencia observacional de términos de T_r.

2.1.- Transformación *EO*.

Teóricamente, para decidir la equivalencia observacional de términos regulares, es suficiente construir para cada término el sistema de transiciones O(S) que representa las relaciones $=\alpha\Rightarrow$ para $\alpha\in P$ y $\Rightarrow$ para la acción τ y decidir la equivalencia por bisimulación entre los sistemas de transiciones construídos (proposition I.2).

Definición II.2.

Si $S=(Q,\cup_{\alpha\in P\tau}\ R_\alpha,\ q_0)$ entonces $\mathcal{O}(S)=(Q,\cup_{\alpha\in P\tau}\ R'_\alpha,\ q_0)$ es el sistema de transiciones que para todo $\alpha\in P$ se verifican las condiciones siguientes:

- $q\xrightarrow{\alpha}q'$ es una transición de $\mathcal{O}(S)$ si y solamente si $q=\alpha\Rightarrow q'$ es una derivación de S
- $q\xrightarrow{\tau}q'$ es una transición de $\mathcal{O}(S)$ si y solamente si $q\Rightarrow q'$ es una derivación de S ($\xrightarrow{\tau}$ es reflexiva en $\mathcal{O}(S)$).

Siguiendo la terminología de Bergstra y Klop [BK 85], llamaremos *saturación* a esta transformación de S en $\mathcal{O}(S)$.

Por ejemplo,si se quiere decidir la equivalencia observacional de los términos u y v definidos como:

u= ((x||y)\d , {x=d.~d.x+a.c.nil , y=~d.(b.nil+d.y)})

v= ((x||y)\d , {x=d.(b.nil+a.~d.nil) , y=~d.d.c.nil})

- Se construyen los sistemas de transiciones S_u y S_v (Figuras S_u et S_v).

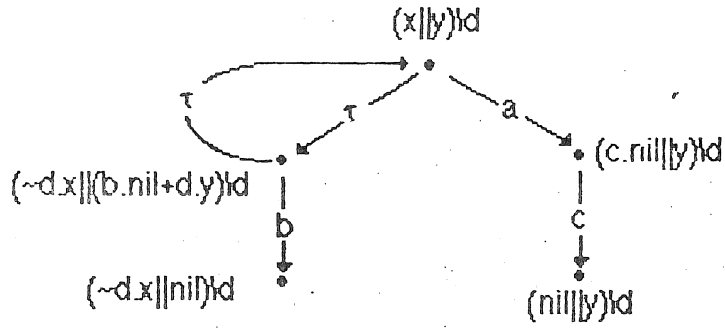


Figura S_u

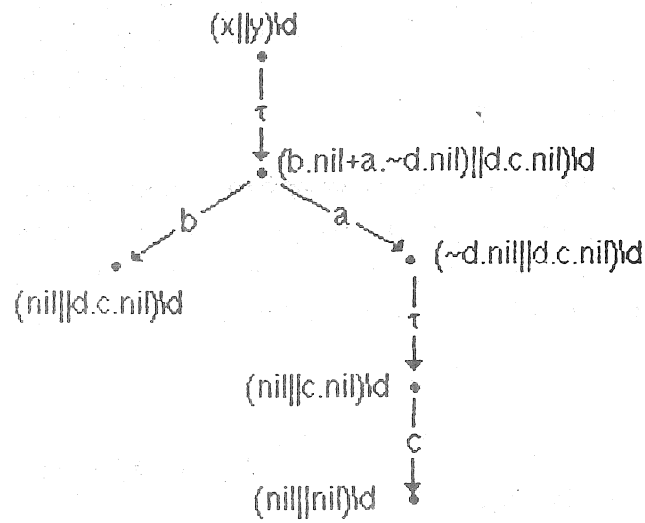


Figura S_Y

- A partir de S_U y de S_Y, se construyen los sistemas de transiciones $\mathcal{T}(S_U)$ y $\mathcal{T}(S_Y)$ (Figura R).
- Se calcula la relación de bisimulación $\mathcal{R}$ entre los sistemas $\mathcal{T}(S_U)$ y $\mathcal{T}(S_Y)$; dicha relación se ilustra en la Figura R. Se concluye que $u \approx v$.

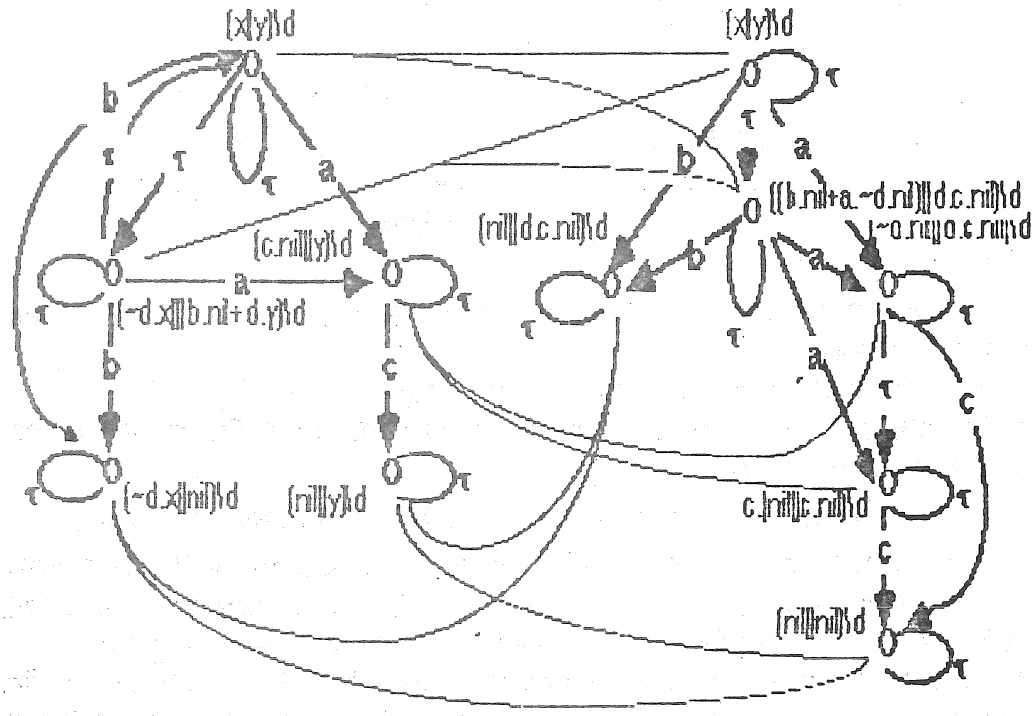


Figura R.

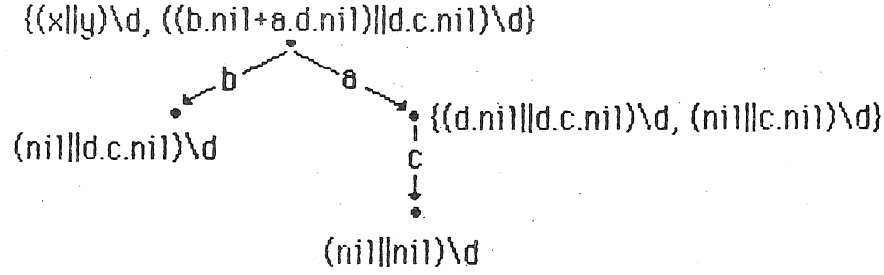
Teniendo en cuenta que la saturación agrega transiciones al sistema, es interesante intentar reducir el sistema inicial antes de la aplicación de dicha transformación.

De hecho, en ciertos casos, es posible de reducir una cantidad considerable de los estados (sub-términos) y de las transiciones (derivaciones) a

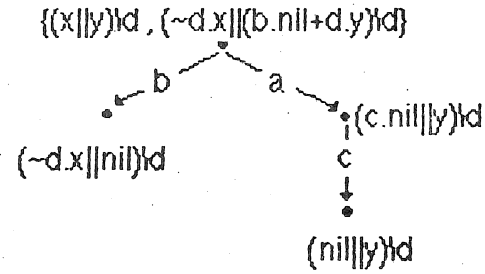
tratar. Estas reduccionespueden definirse de una manera sintáctica a través de reglas de reescritura a aplicarse a los sistemas de transiciones. Estas reducciones permiten en ciertos casos reducir de manera considerable la cantidad de cálculo necesaria para decidir la equivalencia observacional de términos.

Las reducciones que se proponen corresponden a:

- la eliminación de *cadenas* de acciones no observables' (*τ-cadenas*): en el ejemplo anterior, para el sistema de transiciones asociado al término Y, esta reducción produce el sistema de transiciones:



- la eliminación de circuitos de acciones no observables: en el ejemplo anterior, para el sistema de transiciones asociado al término u, esta reducción produce el sistema de transiciones:



De esta manera, para la verificación de la equivalencia observacional del ejemplo anterior:

- se construyen los sistemas de transiciones ilustrados en la Figura R'.
- se calcula la relación de bisimulación R' ilustrada en la Figura R'. Asi es posible concluir que u≈Y.

A continuación, se definen las transformaciones *τ-cadenas* y *τ-circuitos* sobre los sistemas de transiciones asociados a los términos. Enunciamos las propiedades de preservación de la equivalencia observacional. Se define la transformación *EO* y se enuncia la proposición que garantiza la decibilidad de la equivalencia observacional de dos términos por bisimulación aplicando la transformación *EO* a los sistemas de transiciones asociados a dichos términos.

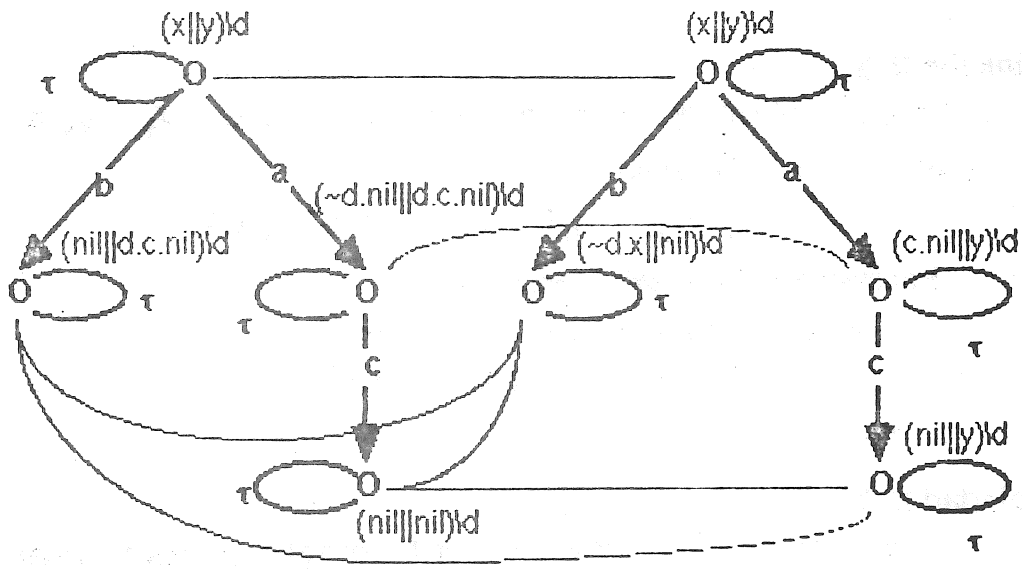


Figura R'

Definición II.3.

Para un sistema de trancisiones $S = (Q, \cup_{\alpha \in P_\tau} R_\alpha, t)$,

- Se dice que una transición $p \xrightarrow{\tau} q$ es una *cadena de* τ en S si y solamente si ella es la única transición posible a partir del estado p.
- Se dice que $p_1 \xrightarrow{\tau} p_2, \ldots, p_n \xrightarrow{\tau} p_1 \in R_\tau$ y $p_1 \xrightarrow{\tau} p_1 \in R_\tau$ constituyen *circuitos de* τ en S.

Propiedad II.1.

Todo par de sistemas de transiciones S_1, S_2 verifica:

$S_1 \approx S_2$ si y solamente si $O(S_1) \sim O(S_2)$.

◆

Definición II.4.

La eliminación de las cadenas de τ de un sistema de transiciones $S_u = (Q, \cup_{\alpha \in P_\tau} R_\alpha, u)$, consiste en:

- eliminar toda transición $t \xrightarrow{\tau} t' \in R_\tau$, si ella es la única transición posible a partir del estado t,
- la substitución del estado t' por el estado t en el sistema resultante.

Propiedad II.2.

La transformación τ-cadenas preserva la equivalencia observacional. Es decir, dos sistemas de transiciones S_1 y S_2 son observacionalmente equivalentes si y solamente si después de la aplicación de la transformación τ-cadenas ellos son observacionalmente equivalentes. ◆

Definición II.5.

Para un sistema de transiciones $S_u = (Q, \cup_{\alpha \in P_\tau} R_\alpha, u)$, la eliminación de circuitos de τ

$$t_1 \xrightarrow{\tau} t_2, \ldots, t_n \xrightarrow{\tau} t_1 \in R_\tau \quad o \quad t_1 \xrightarrow{\tau} t_1 \in R_\tau$$

(transformación *τ-circuitos*) consiste en:

- eliminar toda transición $t_i \xrightarrow{\tau} t_j$ del circuito,
- la substitución de cada estado del circuito por t_1 en el sistema resultante.

Propiedad II.3.

La transformación τ-circuitos preserva la equivalencia observacional.

◆

Définition 1.II.6.

Para todo sistema de transiciones S, la transformación *EO* se define como $EO(S) = O(S')$ donde S' es el sistema obtenido por aplicación de las transformaciones τ−cadenas y τ-circuitos sobre S.

Proposition 1.II.1.

Para todo par de términos t_1, t_2 de T_r

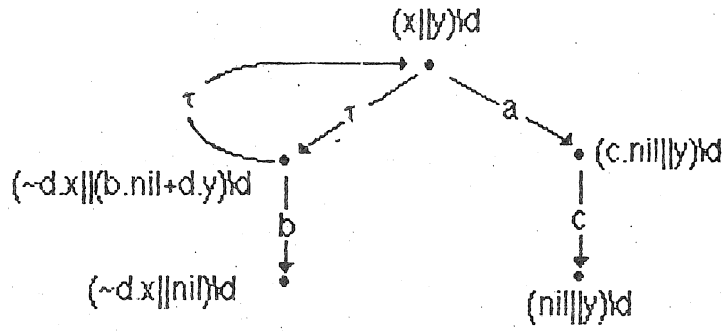
$t_1 \approx t_2$ si y solamente si $EO(S_{t1}) \sim EO(S_{t2})$.

◆

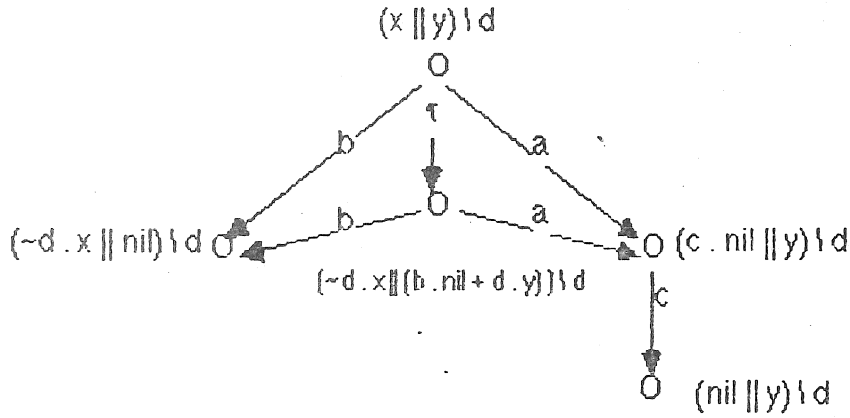
II.2.2.- Transformación *CO*.

De manera similar, se define una transformación (*transformación CO*) sobre los sistemas de transiciones que permite decidir la congruencia observacional por simulación de los sistemas de transiciones obtenidos por aplicación de la *transformación CO*.

Por ejemplo, el sistema de transiciones:



asociado al término u= {((x||y)\d , {x=d.~d.x+a.c.nil , y=~d.(b.nil+d.y))) por aplicación de la transformación *CO*, se transforma en:



Definición II.7.

Para todo sistema de transiciones $S=(Q, \cup_{\alpha\in P_\tau} R_\alpha, q_0)$, la *transformación* $\mathcal{Q}$ procede en tres etapas:

1.- Construcción del sistema de transiciones $S'=(Q', \cup_{\alpha\in P_\tau} R'_\alpha, q'_0)$ por aplicación de las transformaciones τ-cadenas y τ-circuitos sobre el sistema S.

2.- Si durante la construcción del sistema S', se elimina alguna τ-transición que parte del estado inicial q_0, se agrega un nuevo estado inicial $q\notin Q'$, y una transición $q \xrightarrow{\tau} q'_0$ para obtener el sistema de transiciones S".

3.- Transformación de S" en un sistema S"' tal que $\forall\alpha\in P_\tau$ $q\overset{\alpha}{\Rightarrow}q'$ es una transición de S"' si y solamente si $q=\alpha\Rightarrow q'$ es una derivación de S". Cabe indicar que la relación $\xrightarrow{\tau}$ no se hace reflexiva como en el caso de la equivalencia observacional.

Proposición II.2.

La transformación $\mathcal{Q}$ preserva la congruencia observacional.

Prueba.

Por la proposición II.3 y por la definición de la transformación $\mathcal{Q}$, se obtiene:

$S_{t1} \approx S_{t2}$ si y solamente si $\mathcal{Q}(S_{t1}) \approx \mathcal{Q}(S_{t2})$

es decir $\mathcal{Q}$ preserva la equivalencia observacional.

Sea $S_{ti} = (Q_{ti}, \cup_{\alpha\in P_\tau} R_{\alpha\, ti}, q_{io})$ para $i= 1, 2$

$\mathcal{Q}(S_{ti}) = (Q'_{ti}, \cup_{\alpha\in P_\tau} R'_{\alpha\, ti}, q'_{io})$ para $i= 1, 2$

Si en S_{ti} hay una τ-transición $q_{io} \xrightarrow{\tau} q_{ti}$ que forma parte de un circuito o de una cadena de τ, es decir que es eliminada en el paso 1 de la transformación $\mathcal{Q}$, se agrega una transición $q'_{io} \xrightarrow{\tau} q'_{ti}$ en el paso 2, donde $q'_{ti}\in Q'$ es el estado inicial del sistema después de la transformación $\mathcal{Q}$. Así, por

construccción de $\mathcal{CO}(S_{ti})$ para i=1,2 se tiene que:

(1) $q_{io} \xrightarrow{\tau} q_{ti} \in R_{\tau ti}$ implica $\exists q'_{ti} \in Q'_{ti}\ q'_{io} \xrightarrow{\tau} q'_{ti} \in R'_{\tau ti}$ y $q'_{ti} \approx q_{ti}$

$q'_{io} \xrightarrow{\tau} q'_{ti} \in R'_{\tau ti}$ implica $\exists q_{ti} \in Q_{ti}\ q_{io} \xrightarrow{\tau} q_{ti} \in R_{\tau ti}$ y $q'_{ti} \approx q_{ti}$

par (1) y la proposición I.4. se obtiene que $S_{ti} \approx^c \mathcal{CO}(S_{ti})$,

asi, $S_{t1} \approx^c S_{t2}$ si y solamente si $\mathcal{CO}(S_{t1}) \approx^c \mathcal{CO}(S_{t2})$. ◆

Para decidir la congruencia observacional, no es suficiente verificar si los sistemas $CO(S_{ti})$ obtenidos a partir de S_{ti} se bisimulan; en efecto, la relación de transición debe ser reflexiva en algunos casos. Por ejemplo, para decidir la congruencia observacional de los términos b.a.nil y b.(τ.a.nil+a.nil) por bisimulación, es necesario considerar la relación $\xrightarrow{\tau}$ reflexiva. Pero, es fácil ver que si se considera la realción $\xrightarrow{\tau}$ reflexiva, no es posible distinguir por bisimulación los términos a.nil y τ.a.nil; de esta manera, ella no debe ser considerada como reflexiva para el estado inicial. De igual modo, si suponemos la realación $\xrightarrow{\tau}$ reflexiva para todos los estados excepto para el estado inicial, no es posible de verificar por bisimulación que los términos (a.b.x,x=a.b.x) y (x,x=a.b.x) son observacionalmente congruentes.

Por esta razón, introducimos la noción de *r-bisimulación* entre sistemas de transiciones. Esta noción es una variante de la noción de bisimulación; ella se diferencia unicamente por el hecho de que se considera reflexiva la relación $\xrightarrow{\tau}$ de manera asimétrica:

- para el estado inicial q_0 no se agrega jamas un bucle $q_0 \xrightarrow{\tau} q_0$,
- en cada etapa, para la verificación de la condición (i) de la bisimulación, se considera reflexiva la relación $\xrightarrow{\tau}$ únicamente para el segundo sistema de transiciones excepto para el estado inicial e inversamente para la verificación de la condición (ii).

Definición II.8.

Para $S_i = (Q_i, \cup_{\alpha \in P_\tau} R\alpha, q_{i0})$, i=1,2 se dice que S_1 et S_2 *r-bisimulan* (escrito como $S_1 \sim^r S_2$) si y solamente si el mayor punto fijo $R \subseteq Q_1 \times Q_2$ de la funcional F_r contiene el par (q_{10}, q_{20}).

La fonctional F_r se define como:

$F_r(R) = \{(p,q) \mid pRq$ y

y $\forall \alpha \neq \tau$ (si $p \xrightarrow{\alpha} p'$ entonces $\exists q' \in Q_2 (q \xrightarrow{\alpha} q'$ y $p'Rq')$

y si $q \xrightarrow{\alpha} q'$ entonces $\exists p' \in Q_1 (p \xrightarrow{\alpha} p'$ y $p'Rq'))$

y si $p \xrightarrow{\tau} p'$ entonces $((p \neq q_{10}$ y $(p',q) \in R)$ o

$\exists q' \in Q_2 (q = \tau \Rightarrow q'$ y $p'Rq'))$

y si $q \xrightarrow{\tau} q'$ entonces $((q \neq q_{20}$ y $(p,q') \in R)$ o

$\exists p' \in Q_1 (p = \tau \Rightarrow p'$ y $p'Rq')\}$.

Proposición II.4.

Para todo término t_1, t_2 de T_r

$t_1 \approx^c t_2$ si y solamente si $\mathcal{CC}(S_{t1}) \sim^r \mathcal{CC}(S_{t2})$. ◆

CONCLUSION

El objetivo de este trabajo fué el de definir métodos de verificación de sistemas comunicantes descritos en CCS.

Se realizó un estudio del cálculo CCS sin transmisión de valor. Para los términos regulares, se definieron procedimientos de decisión para la congruencia fuerte, la equivalencia observacional y la congruencia observacional. Estos procedimientos de decisión se basan en transformaciones y comparaciones de sistemas de transiciones representando los comportamientos descritos por los términos.

Estos resultados pueden ser aplicados sin muchas variaciones a otros cálculos, como por ejemplo a ACP [BK 84].

REFERENCIAS

[Ab 79] Abrahamson K.
Modal Logic of concurrent non deterministic programs.
Semantics of Concurrent Computation Proceedings.
Lecture Notes in Computer Science 70. Springer-Verlag. 1979.

[AB 84] Austry D., Boudol G.
Algèbre de processus et synchronisation.
TCS 30 pp. 91-131.

[AS 79] Abrial J. R., Schuman S. A.
Non deterministic system verification.
Lecture Notes in Computer Science 70. Springer-Verlag. 1979.

[BH 84] Brookes S. D., Hoare C. A. R., Roscoe A. W.
Theory of communicating sequential processes.
JACM. 31 (3). Juillet 1984.

[BK 84] Bergstra J. A., Klop J. W.
Verification of an alternating bit protocol by means of process algebra.
Centre for Mathematics and Computers.
Departement of Computer Science.
Rapport CS-R8404. Mars 1984. Amsterdam.

[BK 85] Bergstra J. A., Klop J. W.
A complete inference system for regular processes with silent moves.
Centre for Mathematics and Computers.

Departement of Computer Science.
Rapport CS-R8420. Mars 1984. Amsterdam.

[HM 85] Hennessy M., Milner R.
Algebraic laws for nondeterminism and concurrency.
JACM. 32(1). pp. 137-161. Janvier 1985.

[Ko 85] Koomen C.J.
Algebraic specification and verification of communication protocols.
Science of Computer Programming 5. pp 1-36. 1985.

[Mi 80] Milner R.
A calculus of communicating systems.
Lecture Notes in Computer Science 92.
Springer-Verlag. 1980.

[Mi 83] Milner R.
Calculi for synchrony and asynchrony.
T.C.S. 25; pp. 267-310. 1983.

[Mi 85] Milner R.
A complete axiomatisation for observational congruence of finite state behaviors.
Manuscript non publié, décembre 1985.

[Pl 81] Plotkin G.
A structural approach to operational semantics.
Aarhus University. Denmark.
DAIMI FN-19. Septembre 1981.

[Pn 77] Pnueli A.
The temporal logic of concurrent programs.
19th. FOCS. 1977.

[So 87] Soriano A.
YENUS: UN OUTIL D'AIDE A LA VERIFICATION DES SYSTEMES COMMUNICANTES.
Tesis de doctorado. INPG. Francia. 1987.

[SV 85] Soriano A. y Voiron J.
CONCEPCION ET MISE EN OEUVRE D'UN SYSTEME DE PREUVE POUR UN SOUS ENSEMBLE DE CCS SANS TRANSMISSION DE VALEURS.
Panel '85. EXPODATA.
V Congresso da Sociedade Brasileira de Computacao.
XI Conferencia Latino Americana de Informatica. Brasil. 1985.

E D I T O R D E A U T O N O M A S F I N I T O S

FELIPE S. QUADROS ANA M. PRICE
Universidade Federal do Rio Grande do Sul
Porto Alegre - Brasil

R E S U M O

Este artigo descreve um editor gráfico de diagramas de transição desenvolvido com o objetivo de motivar alunos ao estudo de autômatos finitos e auxiliá-los na soluçao de exercícios sobre reconhecedores de linguagens regulares. O editor é de uso extremamente fácil pois sua interface com o usuário compreende, praticamente, uma única tela, na qual estão indicados os modos de operação e os comandos disponíveis ao sistema.

1. INTRODUÇÃO

O editor de automatos faz parte do Laboratório de Compiladores que está sendo desenvolvido para dar suporte às disciplinas relativas ao ensino de compiladores, ministradas em ambos os cursos de Bacharelado e Pós-Graduação em Ciência da Computação da UFRGS. O Laboratório compõe-se, básicamente, de dois módulos:

I) um conjunto de programas que simulam máquinas de Turing e autômatos, cuja finalidade é motivar e auxiliar alunos na solução de exercícios sobre reconhecedores de linguagens tipo 0, 1, 2 e 3 [Hopcroft 69]. Acredita-se que a disponibilidade de simuladores interativos, fáceis de usar e com visual atraente,

servirão de estímulo ao estudo do tema e à solução de exercícios propostos; e

II) um conjunto de subrotinas que implementam algorítmos pertinentes ao processo de compilação, como por exemplo, algorítmos de simplificação de gramáticas, geradores de tabelas LL e LR, analisadores sintáticos, etc. [Aho 77]. Este módulo visa dar suporte aos projetos de compiladores a serem desenvolvidos pelos estudantes. Devido à exiguidade de tempo para implementação e à complexidade de contrução de um compilador, a idéia é que os estudantes desenvolvam alguns módulos do compilador em projeto, integrando-os com módulos disponíveis no Laboratório.

O editor gráfico aqui descrito é um programa interativo, extremamente fácil de ser operado, cuja interface compreende uma única tela e uma dezena de comandos imediatos.

A construção de um autômato finito é efetuada através de operações simples, como por exemplo:

- para criar um estado do autômato, basta movimentar o cursor para a posição desejada na tela e apertar a tecla CR;
- para definir um arco entre dois nodos, devem ser pressionadas as teclas de movimentação do cursor, conforme o caminho desejado, e o arco será traçado.

O capítulo a seguir apresenta fundamentos sobre autômatos finitos. A interface do usuário com o editor é descrita no Capítulo 3. O capítulo seguinte explica como usar o editor para construir autômatos, e o Capítulo 5, apresenta os comandos disponíveis ao usuário. Considerações sobre a atual implementação constituem o Capítulo 6. Conclusões são apresentadas no Capítulo 7.

Segundo [Hopcroft 69], um autômato finito é definido como um sistema M = (K, Σ , ∂, q0, F) onde

K representa um conjunto finito de estados,

Σ é um alfabeto finito de entrada,

∂ é uma função de transição, d: (K X) -> K,

q0 é o estado inicial, e

F é o conjunto de estados finais.

Um autômato finito pode, também, ser representado através de um diagrama de transição em que os nodos representam estados e os arcos, a função de transição. O estado inicial é indicado, explícitamente, por uma seta inclinada, e os estados finais são designados por circunferencias concêntricas. A Figura 1 ilustra um autômato finito que reconhece o conjunto de sentenças formadas por um número par de 0's (zeros) e 1's (uns) [Hopcroft 69].

M = (K, Σ , ∂, q0, F)

Σ = (0, 1)

K = (q0, q1, q2, q3)

F = (q0)

∂ (q0, 0) = q1
∂ (q0, 1) = q3
∂ (q1, 0) = q0
∂ (q1, 1) = q2
∂ (q2, 0) = q3
∂ (q2, 1) = q1
∂ (q3, 0) = q2
∂ (q3, 0) = q0

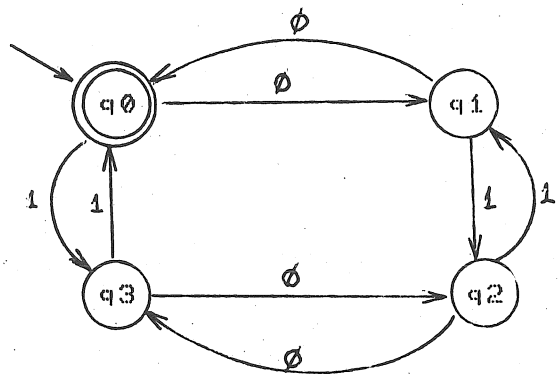


Figura 1

O editor gráfico trabalha com uma representação de autômato prăticamente idêntica à descrita acima, com exceção de que os

estados são denotados por números e o estado inicial deve, obrigatoriamente, ser denotado por 0 (zero). Esta última suposição não impõe quaisquer restrições sobre a representação adotada, mas por outro lado, simplificou a implementação do editor. O autômato exemplo definido acima é mostrado, através da tela do editor gráfico, na Figura 2. (A indicação de estados finais ainda não foi implementada).

3. A TELA DE INTERFACE

A tela de interface com o usuário compõe-se de três partes distintas:

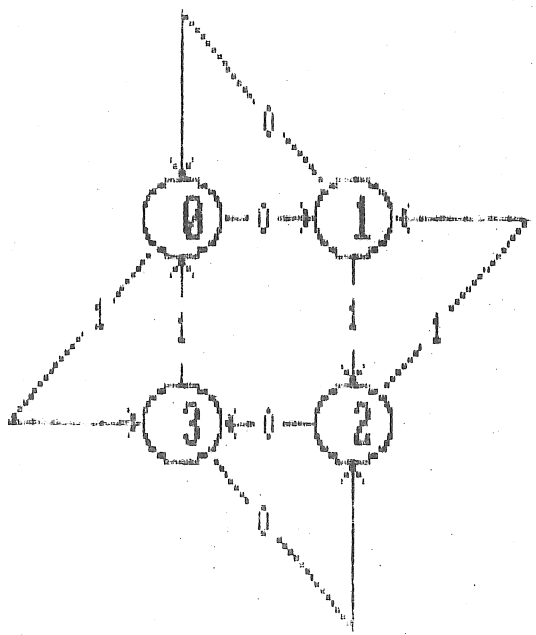
I) a área superior, que ocupa a maior parte da tela, está reservada para o autômato finito em edição. Internamente, esta área é representada como um reticulado de cinco linhas por dez colunas. Os vértices do reticulado indicam posições possíveis para nodos do diagrama, e as arestas e diagonais entre vértices podem conter arcos ou segmentos de arcos do autômato;

II) uma linha para a FITA DE LEITURA; e

III) a área inferior, usada para indicar o modo de operação do editor (EDITANDO, CAMINHANDO, APAGANDO), e mostrar os comandos disponíveis ao usuário, os quais foram implementados através das teclas de função F1 a F10.

4. CONSTRUINDO UM AUTÔMATO

Os comandos do editor (descritos no Capitulo 5) executam funções, tais como, gravar e remover autômato de disco, carregar fita de leitura e listar diretório, etc. A construção efetiva de



FITA DE LEITURA

Modo: Caminhando
Editando
Apagando

F1 - Carrega
F2 - Salva
F3 - Diretorio
F4 - Executa
F5 - Carrega Fita
F6 - Limpa Fita
F7 - Renove
F8 - Imprime
F9 - Apaga
F10 - Fim
ESC - Ajuda

FIGURA 2

um autômato é realizada com o editor funcionando sob um dos modos de operação EDITANDO, CAMINHANDO ou APAGANDO. Quando operando sob um desses modos, as principais teclas para o editor são: CR (Carriage Return), CTRL F (Control F) e as setas de movimentação do cursor (inclusive as setas oblíquas), conforme pode-se perceber pela tabela abaixo. A passagem de um modo para outro é efetuada pela barra de espaço.

MODO DE OPERAÇÃO	CR	SETAS	CTRL F
EDITANDO	CRIA ESTADO OU TROCA NOME DE ESTADO	CRIA ARCO OU TROCA NOME DE ARCO	MARCA ESTADO FINAL
CAMINHANDO		MOVE CURSOR	
APAGANDO	REMOVE ESTADO	APAGA ARCO	DESMARCA ESTADO FINAL

MODO CAMINHANDO:

As teclas CR e CTRL F não produzem efeito algum sobre o autômato em edição, enquanto que, as SETAS deslocam o cursor de um vértice a outro do reticulado. Cabe lembrar que o reticulado é transparente ao usuário.

MODO EDITANDO:

A tecla CR causa a criação de um estado, caso o cursor não esteja posicionado sobre estado algum. Caso contrário, CR faz com que o editor aceite um novo nome para o estado sob o cursor.

As setas de movimentação do cursor geram arcos na direção e

sentido correspondentes, caso o cursor esteja sobre um estado do diagrama e não exista ainda qualquer arco na direção e sentido da SETA pressionada. Se existir tal arco, o editor fica apto a trocar o símbolo indicado no arco.

CTRL F marca o estado sob o cursor como um estado final do autômato.

MODO APAGANDO:

CR causa a extinção do estado sob o cursor se este nodo estiver completamente isolado do diagrama. Isto é, para remover-se um estado do autômato é necessario, em primeiro lugar, apagar todos os arcos que chegam ou saem deste estado. Caso contrário, não ocorre qualquer modificação sobre o autômato.

CTRL F desmarca o estado sob o cursor como estado final do autômato.

As SETAS de movimentação do cursor apagam um arco do diagrama quando o cursor estiver posicionado sobre um estado e existir um caminho na direção e sentido da tecla pressionada. Caso contrário, nao acontece edição alguma sobre o autômato.

5. OS COMANDOS DO EDITOR

Os comandos implementam as funções de ler, gravar, remover autômatos de disco, listar diretório, imprimir diagrama, etc., através das teclas de função (F1 a F10) do teclado. São os seguintes:

CARREGA (F1)

Busca em disco um autômato prèviamente definido e carrega na memória. O nome do autômato é requisitado ao usuário através de

uma janela posicionada à direita da tela. O autômato solicitado é mostrado na tela ficando disponível para execução, edição ou quaisquer outras operações disponíveis no editor.

SALVA (F2)

Armazena em disco o autômato definido na tela, sob o nome especificado pelo usuário, em janela indicada no lado esquerdo da tela (Figura 3).

DIRETÓRIO (F3)

Mostra, em uma janela central, a lista de nomes de autômatos até então catalogados.

EXECUTA (F4)

Inicia o reconhecimento da sentença carregada na FITA DE LEITURA. Cada símbolo da fita ao ser lido fica em vídeo reverso, bem como o estado corrente do autômato em execução. O estado inicial é, por convenção, o estado 0. Os estados finais são indicados por um círculo com duas barras verticais internas. O reconhecimento ou não da sentença em análise é indicado através de mensagem na tela.

CARREGA FITA (F5)

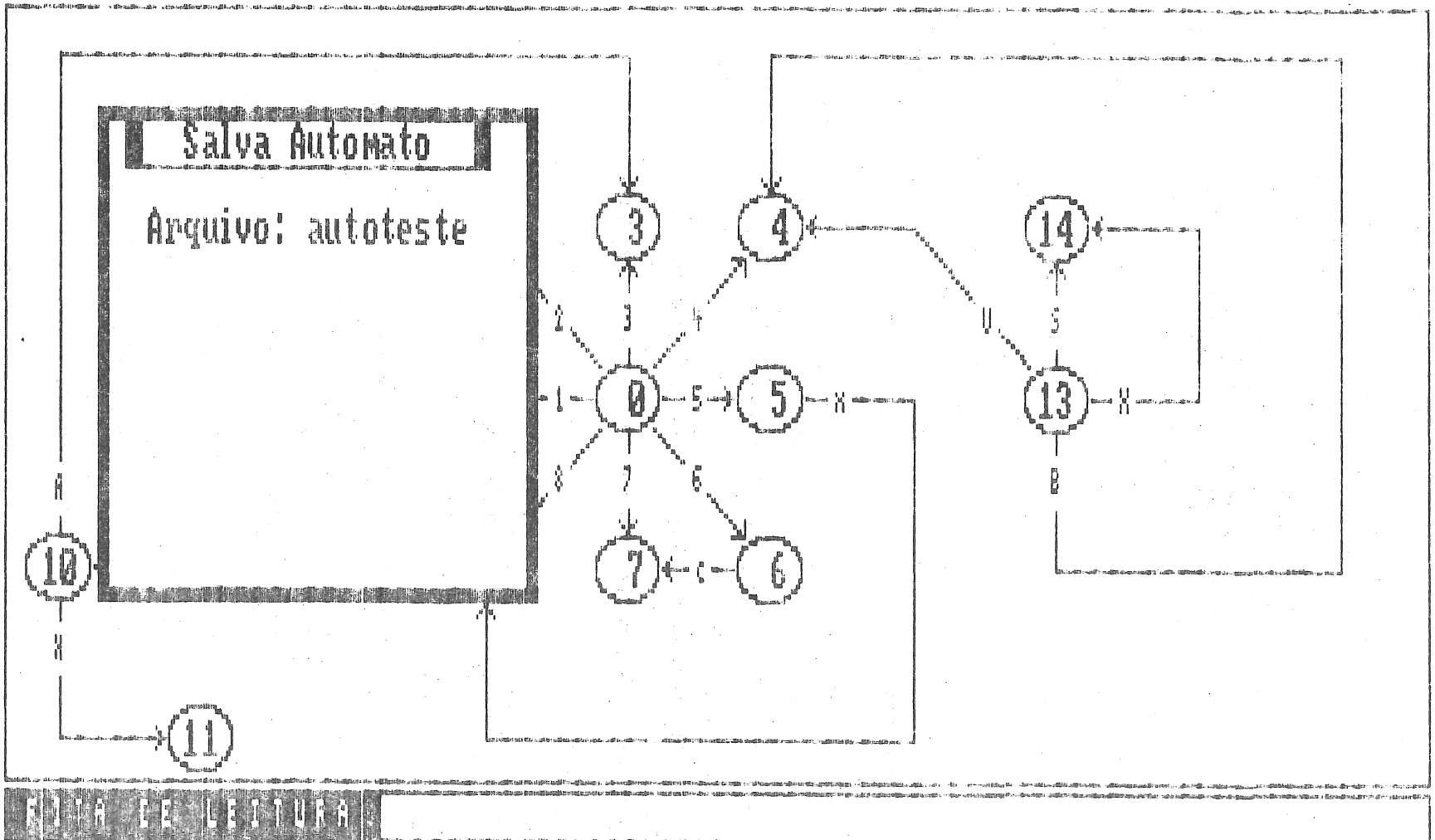
O editor fica apto a receber os símbolos da sentença a ser analisada.

LIMPA FITA (F6)

Preenche a FITA DE LEITURA com espaços.

REMOVE (F7)

Remove do disco o autômato nomeado pelo usuário, através de



Modo: Caminhando
Editando
Apagando

F1 - Carrega
F2 - Salva
F3 - Diretorio
F4 - Executa
F5 - Carrega Fita
F6 - Limpa Fita
F7 - Remove
F8 - Imprime
F9 - Apaga
F10 - Fim
ESC - Ajuda

FIGURA 3

uma janela central (superior).

IMPRIME (F8)

Produz, na impressora, uma cópia do autômato mostrado na tela.

APAGA (F9)

Apaga o autômato mostrado na tela.

FIM (F10)

Encerra a sessão.

AJUDA (ESC)

Mostra na tela o manual do usuário.

As janelas de mensagens são distribuidas na tela em posições distintas para que o usuário associe a posição da janela com a função solicitada.

6. CONSIDERACOES SOBRE A IMPLEMENTACAO

O editor foi desenvolvido prăticamente em Turbo Pascal, em microcomputadores do tipo PC, contendo, porém, algumas rotinas de manipulação grafica em Assembler para aumentar o desempenho do sistema. O código fonte contém atualmente 1900 linhas.

As funções CARREGA FITA, EXECUTA e CTRL F (marca e desmarca estado final) ainda não estao disponíveis na atual implementação. Por outro lado, a edição de autômatos está completamente operacional. A implementação atual ainda não permite a escrita de

símbolos alternativos sobre arcos.

7. CONCLUSÕES

Acredita-se que a disponibilidade de ferramentas interativas de apoio ao ensino estimulam o estudo do tema associado e auxiliam a solução de problemas propostos em classe. O editor gráfico de autômatos finitos descrito neste relatório foi desenvolvido com esse objetivo.

O editor implementado é um programa interativo e fácil de ser operado devido à simplicidade de sua interface com o usuário: uma única tela, onde estão indicados os modos de operação e os comandos disponíveis, os últimos implementados através de teclas de função.

A construção de um autômato é realizada básicamente com o auxílio das teclas CR, CTRL F e as setas de movimentação do cursor. Imediatamente após a criação de um estado ou arco de autômato o editor exige a identificação do elemento criado, mantendo pois o diagrama sempre consistente.

O editor oferece, também, facilidades para gravação, impressão e catalogação de autômatos, e futuramente, a possibilidade de consulta ao manual do usuário.

Pretende-se introduzir esta ferramenta como suporte ao ensino de reconhecedores de linguagens regulares nos cursos de graduação e pós-graduação no próximo semestre letivo.

AGRADECIMENTOS

Os autores agradecem os incentivos recebidos do Conselho

Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

REFERÊNCIAS BIBLIOGRÁFICAS

[Hopcroft 69]

J. Hopcroft & J. Ullman, Formal Languages and Their Relation to Automata, Addison-Wesley, 1969.

[Aho 77]

A. Aho & J. Ullman, Principles of Compiler Design, Addison-Wesley, 1977.

S. Carvalho, Turbo Pascal, Version 3.0 Reference Manual, Borland International, 1985.

Los Conjuntos Difusos en la Modelización Linguistica

Francesca Greselin Ferrari

Laboratorio de Electrónica Digital
Universidad Católica Nuestra Señora de la Asunción
Asunción - Paraguay

El análisis de muchos problemas del mundo real necesita el manejo de tipos diferente de informaciones: unos específicos, asertados, y otros vagos, con contornos imprecisos. Los expertos tienen entonces que formular juicios y reglas heuristicas, calificandolos con descriptores linguisticos como "considerable", "posible", "importante"...

El desarrollo de los Sistemas Expertos necesitó enfocar desde su comienzo el problema de modelar informaciones y relaciones verisimiles aunque no precisas. El ambiente matematico que pudo permitir la introducción de la incertitumbre en la definición de objetos y reglas es la teoría de los conjuntos difusos.

En este articulo se presenta un rápido recorrido en la teoría y una tecnica que permite hallar una solución para las operaciones entre conjuntos difusos. Todo esto permite modelar el lenguaje natural á través de variables linguisticas. Con estas ultimas se provee al final una aplicación para previsiones de poblaciones estudiantiles.

INTRODUCCION

El desarrollo de los sistemas expertos empezó su éxito previendo la conducta y dando juicios en campos de la realidad que no pueden ser modelados de manera determinística.

Muchas situaciones reales, en efecto, no permiten partir de **datos ciertos** y/o no pueden ser juzgados con reglas expresadas en **términos matemáticos convencionales,** ya sean deterministicos o probabilisticos.

La información, en general, viene de datos, mediciones o experiencias precedentes. Entonces puede ser vaga, imprecisa, cualitativa, linguística o incompleta. Los expertos, por lo tanto, además de modelos deterministicos, tienen juicios basados en reglas heuristicas que les permitan lograr previsiones en casos en los cuales hayan factores muy difíciles de medir o cuantificar precisamente. Médicos, psicologos, ingenieros, manejan informaciones que frecuentemente están acompañadas por calificativos como "serio", "importante", "considerable" y de estos datos realizan evaluaciones a través de procedimientos empíricos, ill-structured.*

La incertidumbre puede ser dividida en tres clases (según lo especificado en el " Workshop on Civil Engeneering Applications of Fuzzy Sets" , Purdue University, Octubre 1985):

- casualidad natural que causa incertidumbre

- imprecisión que causa difusión **

- ignorancia.

La teoría de las probabilidades maneja matemáticamente el primer tipo de incertidumbre y trata de reducir la difusión al mínimo. Pero, en todo caso, los métodos

---

* Una regla "ill-structured" es aquella donde la relación entre antecedentes y consecuentes no ha sido suficientemente estudiada o para la cual no hay disponible un modelo matemático razonablemente bueno. En este caso, se pueden usar reglas del tipo IF-THEN para expresar una relación suelta entre antecedentes y consecuentes.

** En este artículo entenderemos 'difusión' como 'fuzziness' y 'difuso' como 'fuzzy'.

probabilísticos están basados en la teoría clásica de los conjuntos, en la cual una proposición o es verdadera o es falsa. Muchas de las proposiciones comunes no son fácilmente clasificables en tal contexto. La teoría de los conjuntos difusos ("fuzzy sets") permite enfrentar y tratar aquellas clases de objetos del mundo real que son **'difusas'** por naturaleza, en el sentido que la **transición entre la pertenencia y la no pertenencia a estas clases es gradual más que brusca.**

De hecho, dado un objeto x y una clase F, la pregunta real en muchos casos no es si x es o no miembro de F, sino **el grado en que x pertenece a F** o, en forma equivalente, el nivel de pertenencia de x a F.

Sea, por ejemplo, U un conjunto de personas:

U = {Alberto, Juana, Ricardo, Manuela, Pablo}

Si queremos caracterizar el subconjunto A de U que recoja las personas altas de estatura, intuimos que es claro incluir Alberto -que mide 1.92 cm de estatura-, pero es más difícil juzgar en que medida Juana -1.63 cm de estatura-, pertenezca a A.

Cuando los conjuntos difusos son utilizados para modelar informaciones difusas, el análisis implica y necesita de operaciones algebricas extendidas, es decir, **operaciones entre números difusos.**

En este artículo haremos, entonces, un rápido recorrido en los **conceptos básicos de la teoría de los conjuntos difusos.** En detalle veremos después una técnica basada en la representación de conjuntos difusos a través de α-cortes y de un análisis combinatorio de intervalos. Este método permite hallar una **solución discreta pero exacta para las operaciones entre conjuntos difusos** de una manera muy eficiente y simple.

Presentaremos la **conversión entre variables linguísticas** (del lenguaje natural) y **variables numéricas** (conjuntos difusos).

Posteriormente, como ejemplo, veremos una **aplicación** para la modelización de estudios en el campo educativo y de paso vamos a presentar también el 'modelo en cascada' para analizar problemas complejos.

En conclusión, se dará también una breve descripción del papel que el procesamiento de informaciones difusas tiene en el campo de los **sistemas expertos.**

BASES DE LA TEORIA DE LOS CONJUNTOS DIFUSOS

Sea U un universo, o sea una colección arbitraria de objetos, conceptos o costrucciones matemáticas.

Convencionalmente, si A es un subconjunto de U cuyos elementos son $u_1, ..., u_n$, entonces A es expresado como:

$$A = \{u_1, ..., u_n\}.$$

Para nuestros propósitos, sin embargo, es mas conveniente expresar A como la unión de los u_i:

$$A = u_1 + ... + u_n = \sum_{i=1}^{n} u_i$$

teniendo en cuenta que, por cada i, j:

$$u_i + u_j = u_j + u_i$$

y:

$$u_i + u_i = u_i \ .$$

Como una extensión de esta notación, Zadeh (1964) sugirió el concepto de conjunto difuso F de U expresandolo mediante:

$$F = \mu_1 u_1 + ... + \mu_n u_n$$

o, equivalentemente, como:

$$F = \mu_1/u_1 + ... + \mu_n/u_n$$

donde μ_i, por $i = 1, ..., n$, representan el grado de pertenencia de los u_i a F.

A menos que no se establezca lo contrario, los μ_i se asumen con valores en el intervalo [0,1], con 0 y 1 que denotan 'no pertenencia' y 'plena pertenencia' respectivamente.

El papel que en la teoría de conjuntos clasicos jugaba la función característica (midiendo en forma binaria la pertenencia de cada elemento al conjunto A) :

$$\chi_A(x) = \begin{cases} 1 & \text{si x pertenece a A} \\ 0 & \text{si x no pertenece a A} \end{cases}$$

está ampliado y generalizado a través de la **función de pertenencia** en la teoría de conjuntos difusos, permitiendo describir **niveles diferentes** de pertenencia de cada elemento de F.

La representación de un conjunto difuso finito es una forma lineal en los u_i.

Consistentemente , un subconjunto difuso arbitrario de U puede ser expresado mediante la integral:

$$F = \int_U \mu_F(u) / u$$

donde:

$$\mu_F : U \dashrightarrow [0,1]$$

es la función de pertenencia de F, y la integral en U denota la unión de los elementos difusos $\mu_F(u) / u$ en el universo U.

Ejemplo:
Sea X el conjunto de los enteros no negativos, $X = \mathbb{Z}^+$, y sea F el subconjunto difuso de X de los "enteros pequeños". Definimos la función de pertenencia de F como:

$$\mu_{\text{enteros pequeños}}(x) = \{ 1 + (x/5)^2 \}^{-1} \qquad x = 0, 1, 2, ...$$

OPERACIONES ENTRE CONJUNTOS DIFUSOS

Sean F y G dos conjuntos difusos en el universo U.

Sean $\wedge$ y $\vee$ los operadores de mínimo y de máximo respectivamente.

Se define la **unión**:

$$F \cup G = \int_U \{ \mu_F(u) \vee \mu_G(u) \} / u$$

la **intersección**:

$$F \cap G = \int_U \{ \mu_F(u) \wedge \mu_G(u) \} / u$$

el **complemento**:

$$\overline{F} = \int_U \{ 1 - \mu_F(u) \} / u$$

Se muestra fácilmente que valen las identidades:

$$\overline{(F \cup G)} = \overline{F} \cap \overline{G}$$

y

$$\overline{(F \cap G)} = \overline{F} \cup \overline{G}$$

En el mismo contexto, se define que F contiene G si i solo si:

$$F \supseteq G \quad <===> \quad \mu_F(u) \geq \mu_G(u) \qquad \forall \; u \in U$$

Si $F_1, ..., F_n$ son subconjuntos difusos de $U_1, ..., U_n$, entonces el **producto cartesiano** de $F_1, ..., F_n$ es un subconjunto difuso de $U_1 x \; ... \; x U_n$, definido por:

$$F_1 x \; ..... \; x F_n = \int_{U_1 x ... x U_n} [\mu_F(u_1) \wedge \; ..... \wedge \mu_F(u_n)] \; / \; (u_1, ..., u_n)$$

PRINCIPIO DE EXTENSION

Sea F un conjunto difuso definido en X y sea $y = f(x)$. La imágen de F bajo f se define como el conjunto difuso G definido en Y y dado por:

$$G = \int_U \mu_G(y) \; / \; y$$

donde:

$$\mu_G(y) = \mu_F(x) \; | \; y=f(x)$$

La operación de mapeo puede ser extendida al caso general y esto se conoce como el **principio de extensión** (Zadeh 1973).

Sean $F_1, F_2, ..., F_n$ conjuntos difusos definidos en $X_1, X_2, ..., X_n$ respectivamente. Sea f una función:

$$f: \; X_1 \; x \; X_2 \; x \; ... \; x \; X_n \longrightarrow Y$$

la imágen difusa G de $F_1, F_2, ..., F_n$ bajo f tiene como función de pertenencia:

$$\mu_G(y) = \bigvee_{\substack{x_1 \in X_1, ..., x_n \in X_n \\ y = f(x_1, ..., x_n)}} [\wedge (\mu_A(x_1), ..., \mu_A(x_n))]$$

Todas las relaciones matemáticas entre elementos no difusos, y evidentemente también las operaciones algebraicas, pueden ser extendidas a través de este principio en los conjuntos difusos.

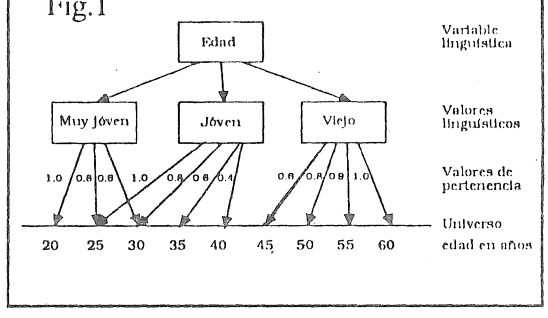
Quedan así definidas las **operaciones extendidas**.

EL LENGUAJE NATURAL

La creación de los conjuntos difusos abrió el camino para investigar el lenguaje natural, cuya 'difusión' era conocida desde hace mucho tiempo, permitiendo que este mismo lenguaje pueda ser usado en procedimientos analíticos.

Informalmente, una variable linguística es una variable cuyos valores -en vez de ser números- son palabras o frases del lenguaje natural (y análogamente se pueden definir para cualquier lenguaje artificial).

Fig.1



Por ejemplo, si "edad" es interpretada como una variable linguística, entonces el conjunto de sus valores linguísticos, llamado "term-set", en este caso T (edad), puede ser imaginado como

el que recoje toda expresión del tipo: "jóven", "viejo", "muy jóven", "no muy jóven", "muy muy viejo", "mas o menos jóven", "bastante jóven"...

Cada uno de estos términos es una etiqueta de un conjunto difuso en el universo de las edades, es decir, U = [0, 100].

A una variable linguística L se le asocian dos reglas:

a) una regla **sintactica** que define las sentencias bien estructurada en T(L)

b) una regla **semantica**, a través de la cual el sentido de los términos en T(L) puede ser definido.

Si t es un término en T(L), entonces su significado es un subconjunto de U.

Un **término primario** en T(L) es un término cuyo significado tiene que ser definido a priori, y que sirve como base para computar el sentido de los términos no-primarios en T(L).

Por ejemplo, en T(edad), los términos primarios son "viejo" y "jóven", cuyos significados pueden ser definidos por sus funciones de pertenencia $\mu_{\text{jóven}}$ y μ_{viejo}:

$$\text{"viejo"} = V = \int_U \mu_{\text{viejo}}(u) / u$$

$$\text{"jóven"} = J = \int_U \mu_{\text{jóven}}(u) / u$$

Los términos primarios o atómicos pueden ser conectados entre sí con los conectores linguísticos "y" (conjunción), "o" (disyunción) -identificables con la operación de intersección y de unión respectivamente- y "no". Además, pueden ser modificados con especificadores linguísticos como "muy", "ligeramente", "mas o menos" ...

Después de haber cuantificado y modelado los términos primarios con los conjuntos difusos, todos los términos compuestos y modificados pueden ser definidos a través de las operaciones extendidas .

Podemos definir :

$$\text{"no jóven"} = J = \int_U [1 - \mu_J(u)] / u$$

$$\text{"muy jóven"} = J^2 = \int_U [\mu_J(u)]^2 / u$$

$$\text{"mas o menos jóven"} = J^{1/2} = \int_U [\mu_J(u)]^{1/2} / u$$

Se puede también hacer el camino al revés, o sea la conversión entre un valor numérico y un término linguístico, simplemente comparando la función de pertenencia obtenida por medios computacionales y la que más se le acerque entre las de los términos primarios conocidos.

METODO DE LOS VERTICES

El principio de extensión prevee una solución teórica general al algebra de conjuntos difusos. Sin embargo, hallar la solución en muchos casos reales es tarea muy ardua, aún con la ayuda de una computadora.

Bass & Kwakernaak buscaron lograr una solución analítica reconociendo en lo ante dicho un problema de programación no-lineal, pero requiriendo condiciones muy restrictivas y con una implementación bastante engorrosa.

Otros hallaron soluciones aproximadas (Dubois & Prade), (Schmucker, Brown) con métodos de discretización y encontraron problemas a causa de la naturaleza discontinua de las operaciones de máximo y mínimo.

Un nuevo enfoque para evaluar la función de pertenencia fue presentado por Dong y Wong (1985) y se llama **método de los vértices**.

Sea entonces:

$$y = f(X_1, X_2, ..., X_n)$$

una función donde los antecedentes X_1, X_2,..., X_n son conjuntos difusos y se quiera estimar el consiguiente output conjunto difuso Y.

Para este fin se define el α-corte del conjunto difuso (o conjunto de nivel α de X_i), como:

$$X_i{}^{\alpha} = [x \mid \mu_X(x) \geq \alpha] \qquad \text{con } \alpha \in [0,1]$$

y también el α corte de Y como:

$$Y^{\alpha} = f(X_1{}^{\alpha}, X_2{}^{\alpha}, ..., X_n{}^{\alpha})$$

donde los $X_i{}^{\alpha}$ son n intervalos que forman una región rectangular n-dimensional con 2^n vértices.

Se puede demostrar facilmente que si la función f es continua y monótona, entonces:

$$Y^{\alpha} = [\min_j \{f(c_j)\}, \max_j \{f(c_j)\}] \qquad j = 1, ..., 2^n$$

donde c_j es la coordenada del j-esimo vértice.

Necesitamos ahora del concepto de modelo en cascada para poder ilustrar después con un ejemplo de aplicación lo que se ha dicho hasta ahora.

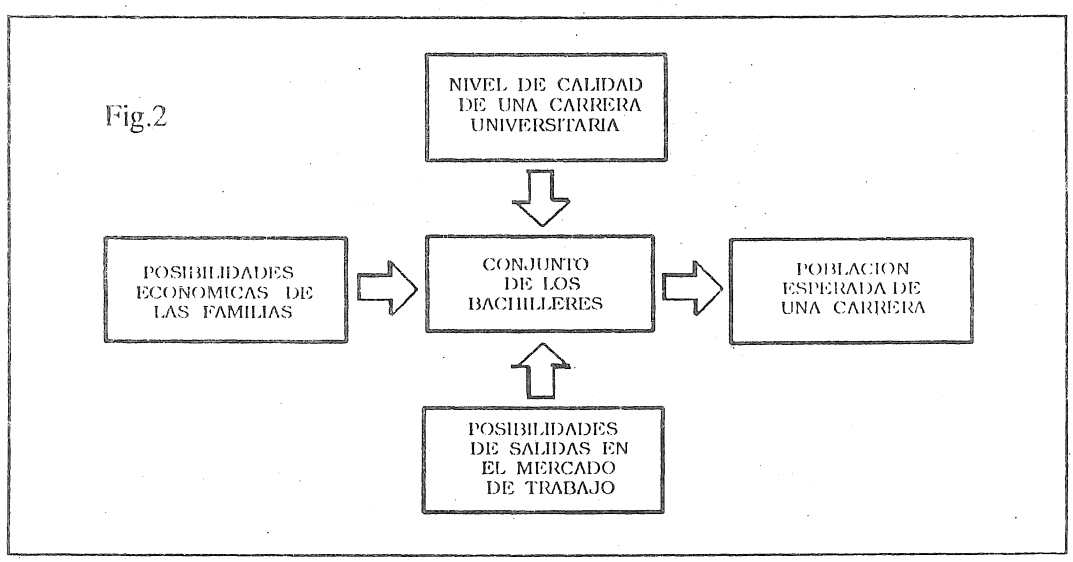
MODELO EN CASCADA

Según el punto de vista bayesiano, hay muchas situaciones reales en las cuales las nuevas informaciones pueden ser utilizadas para actualizar las estimaciones precedentes.

Supongamos que se quiera modelar la expresión linguística siguiente:
"Si el conjunto de los bachilleres tiene una preparación bastante amplia, y se tiene en cuenta que la posibilidad de las familias para mantener sus hijos en los estudios es medianamente buena, se puede razonablemente esperar una respuesta alentadora en un medio universitario de nivel interesante, considerando que también hayan buenas posibilidades laborales en perspectiva".

Un esquema lógico efectivo podría ser el siguiente:

Fig.2



Este esquema resulta complejo porque cada característica que influye en el proceso está independientemente considerada para la evaluación del resultado final, o sea para la evaluación de la población esperable en una carrera.

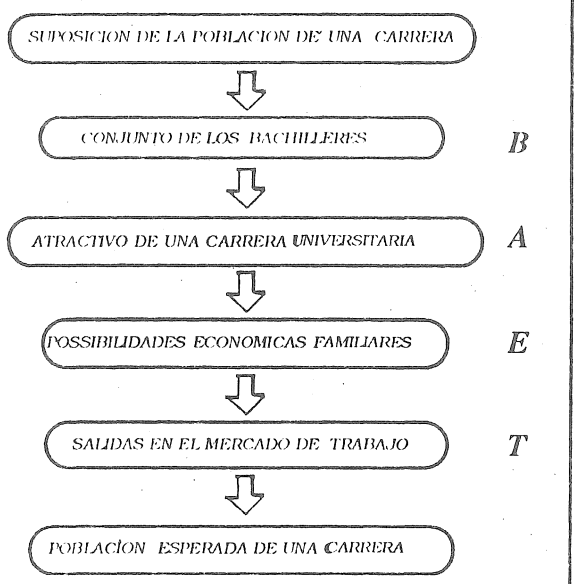
Para modelar en términos de conjuntos difusos todo este esquema, no tendríamos problemas en describir cada antecedente mediante una función de pertenencia razonablemente adecuada, es decir, mediante una representación de conjuntos difusos. Las "flechas" del dibujo

serían nuestras reglas ill-structured, o sea nuestras reglas difusas. Lo que resulta más difícil de vislumbrar es cómo tener en cuenta **simultáneamente** todos los factores, mediante una regla unificadora.

El procedimiento de evaluación de la población esperada de una carrera puede ser pensado como una **sucesión de decisiones, mientras más y más datos son incorporados en la estimación** - de manera parecida al análisis bayesiano - para actualizar una estimación precedente con nuevas informaciones. Esto permite representar una evaluación compleja a través de una simple serie lineal de operaciones, que parte de una suposición inicial de evaluación.

Apliquémoslo al ejemplo anterior:

Fig.3



Sea entonces X el conjunto difuso que representa la población hipotética de una carrera universitaria y sea Y la evaluación que queremos lograr. Y puede ser obtenido mediante las sucesivas actualizaciones expresadas como:

$$Y = \alpha T * \beta E * \chi \Lambda * \delta B * X$$

suponiendo que T, E, Λ, y B sean reglas difusas determinadas por las afirmaciones linguísticas, que α, β, χ, δ sean pesos a través de los cuales queremos dar diferente importancia a los cuatro factores T, E, Λ, B asumidos independientes entre ellos, y que ' * ' sea un operador que indica una relación difusa.

Siguiendo siempre el mismo ejemplo, queremos presentar una aplicación del método de los vértices. Suponemos haber medido que la población X de los bachilleres de una ciudad sea dividida en tres partes iguales, según los diferentes niveles de preparación hallada:

X = {estudiantes con preparación de nivel muy bajo} +

+ {estudiantes con preparación de nivel alto} +

+ {estudiantes con preparación de nivel bueno}

'Muy bajo', 'alto', 'bueno' son términos linguísticos, cuya representación por conjuntos difusos es :

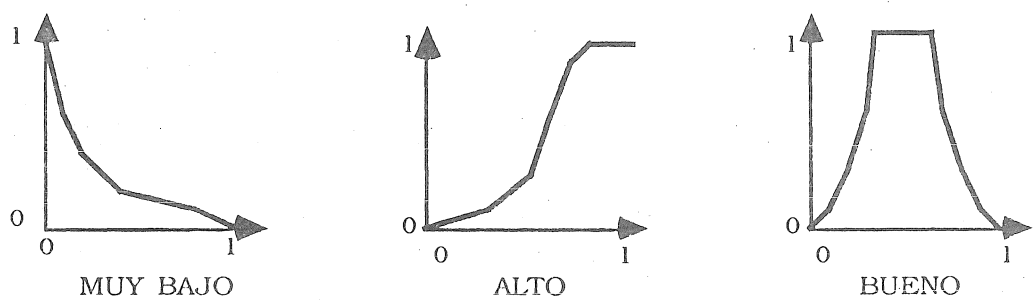


Fig.4

Juzgamos que sea razonable pensar que frente a la posibilidad de cursar -supongamos- una carrera técnica de nivel interesante, sea muy baja (W_1) la inscripción entre los estudiantes de nivel inferior (X_1), a causa de las pretensiones de la carrera misma. Habrá alta elección (W_2) de la carrera entre estudiantes de buen nivel (X_3) y se supone una media participación (W_3) entre los de alta preparación (X_2) (que pensamos tendrán intereses mejor definidos que los precedentes).

La estimación es entonces una función de las tres componentes de población diferentes por nivel de preparación- pesadas mediante W_1, W_2, W_3.

Se reconoce que:

W_1 = 'muy bajo' = X_1

W_2 = 'alto' = X_3

W_3 = 'medio' = X_2

Los tres componentes de población X_i , com también los pesos W_i son conjunto difusos ; la población resultante será el conjunto difuso que se halla a través de la media pesada:

$$Y = \frac{\Sigma_i \; W_i \, X_i}{\Sigma_i \; W_i}$$

Entonces, de las dos ultimas relaciones obtenemos:

$$Y = f(X_1, X_2, X_3) = \frac{X_1^2 + 2\, X_2 X_3}{X_1 + X_2 + X_3}$$

Eligendo $\alpha = 0{,}6$ resultan los siguientes α-cortes de los X_i :

$X_1^{0,6}$ = [0 , 1/10]

$X_2^{0,6}$ = [3/5, 1]

$X_3^{0,6}$ = [3/10, 7/10]

así que los 2^3 vértices resultantes son:

c_1 = (0, 3/5, 3/10) c_2 = (1/10, 3/5, 3/10)

c_3 = (0, 1, 3/10) c_4 = (1/10, 1, 3/10)

c_5 = (0, 3/5, 7/10) c_6 = (1/10, 3/5, 7/10)

c_7 = (0, 1, 7/10) c_8 = (1/10, 1, 7/10)

Podemos ahora hallar las $f^{0,6}(c_i)$, obteniendo:

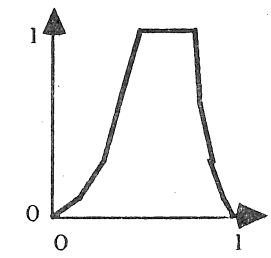
$f^{0,6}(c_i)_{i=1,\ldots,8}$ = { 2/5, 37/100, 6/13, 61/140, 42/65, 17/28, 14/17, 47/60 }

entonces:

$Y^{0,6}$ = [37/100, 14/17]

Análogamente, calculando Y^{α} para diferentes valores de $\alpha \in [0, 1]$, se logra el gráfico de Y:

Fig.5



POBLACION RESULTANTE

La población hallada puede entonces ser referida a una población universitaria de 'medio' nivel de preparación (compare este ultimo gráfico con el de X_3, y haga la conversión de conjunto difuso a variable linguística).

CONJUNTOS DIFUSOS Y SISTEMAS EXPERTOS

Ya dijimos que gran parte del conocimiento, especialmente el ganado de la observación, es de alguna manera probabilístico y muchas veces también incompleto. Eso significa que se puede conciderar algo como verdadero, mientras que aún se permita la posibilidad de un error. Para que una computadora pueda pensar y razonar como un ser humano, se precisa entonces una integración de esta incertidumbre con la lógica.

La mayoría de los hombres manejan la incertidumbre que viene de la observación natural sin muchos problemas, porque están acostumbrados a ella. Sin embargo, hacer posible que una computadora sepa trabajar con la incertidumbre, necesita de un buen esfuerzo. En el lenguaje de los programadores, es una tarea no banal la de juntar la lógica con la incertidumbre, porque son opuestas.

La lógica difusa nos permite pensar en valores verdaderos en el sentido, más debil pero mucho más próximo a la realidad, de ser verosimiles.

Es facil intuir como, para los problemas de reconocimiento y clasificación de patrones (pattern recognition y pattern classification) y para los sistemas expertos, el manejo de la incertidumbre vuelve a ser un punto crítico de evaluación de las posibilidades de la Inteligencia Artificial.

Se puede subdividir el problema en dos pasos distintos:

a) convertir el dato p en un objeto matemático $M(p) = \{ M_1(p), ..., M_n(p) \}$

b) pasar de algoritmos 'opacos' A_{op} a algoritmos 'transparentes' A_{tp}, o sea de algoritmos cuyas descripción no está todavía conocida, a algoritmos que actúen sobre el conjunto M(p) realizando el grado de pertenencia de p como definido por A_{op}.

Es importante observar que desde un punto di vista práctico, es deseable que:

a) M(p) sea definido por un pequeño número de atributos

b) la medición de estos atributos sea relativamente simple.

En la casi totalidad de los Sistemas Expertos, si se desea seguir la formulación descripta, se encuentran problemas para lograr una buena definición matemática clásica.

Ya vimos que los métodos de la lógica difusa se pueden aplicar ya sea en la evaluación de los datos, como en la costrucción de reglas 'sueltas'. La estructura conceptual de la teoría de los conjuntos difusos provee entonces un ambiente más natural para la formulación y la solución aproximada de estos tipos de problemas, mucho más apta que los métodos tradicionales , basados en la teoría clásica de los conjuntos, en la teoría de las probabilidades y en la lógica booleana.

Bibliografia

Baas S. and Kwakernaak H. (1979), *"Rating and Ranking of Multiple Aspects Alternatives Using Fuzzy Sets"*. Automatica, 13.

Brown C.B. and Yao J.T.P. (1983), *"Fuzzy Sets and Structural Engineering"*. ASCE J. Struct. Eng., 109, ST5.

Dong W.M. and Shah H.C. (1985), *"Vertex Method for Computing Function of Fuzzy Variables"*, Journal of Fuzzy Sets and Systems.

Dubois D. and Prade H. (1980), *"Fuzzy Sets and Systems: Theory and Applications"*. Academic Press, New York.

Duda R., Hart P. and Nilsson N. (1976), *"Subjective Bayesian Methods for Rule-based Inference Systems"*. Proceedings of National Computer Conference, pp. 1075-1082.

Kaufmann A. and Gupta M.M. (1985), *"Introduction to Fuzzy Arithmetic: Theory and Applications"*. Van Nostrand Reinhold Company, New York.

Schmucker K.J. (1984), *"Fuzzy Sets, Natural Language Computational, and Risk Analysis"*. Computer Science Press, Potomac, Maryland.

Wong F.S., Dong W., Boissonnade A. and Ross T.J. (1986) *"Expert Opinions and Expert Systems"*. ASCE 9th Electronic Computations Conf., Birmingham, Alabama.

Zadeh L.A. (1964), *"Fuzzy Sets"*, J. Fuzzy Sets and Systems, Memo, EREL., No. 64-44, University of California, Berkeley.

EL DICCIONARIO ELECTRONICO DEL ESPAÑOL

CARLOS SUBIRATS RÜGGEBERG

UNIVERSIDAD AUTONOMA DE BARCELONA
FACULTAD DE LETRAS
DEPARTAMENTO DE FILOLOGIA ESPAÑOLA
08193 Bellaterra (Barcelona), ESPAÑA

RESUMEN

El diccionario electrónico consiste en una lista exhaustiva del léxico español; cada una de las entradas de dicha lista va acompañada de una codificación alfanumérica, que es utilizada por el paquete de programas de generación automática de las formas flexivas. Dichos programas utilizan la información del código alfanumérico para generar todas las formas flexivas; a su vez, para cada una de las formas generadas, los programas especifican:

a. las principales propiedades morfológicas de flexión;

b. su relación formal con la forma de base del diccionario electrónico.

La posibilidad de disponer, por primera vez, de un léxico codificado y exhaustivo tanto de las formas de base como de las formas flexivas de la lengua española, permitirá realizar tanto investigaciones teóricas como aplicaciones tecnológicas de carácter radicalmente nuevos.

EL DICCIONARIO ELECTRONICO DEL ESPAÑOL(1)

1. CARACTERISTICAS GENERALES

1. La lingüística y la informática se han unido en los últimos años en un campo de investigación interdisciplinar que ha alcanzado un importante desarrollo en las investigaciones centradas en la simulación por medios informáticos de aquellos aspectos del comportamiento humano en los que interviene el lenguaje natural. Las investigaciones conjuntas de la informática y la lingüística no han girado en torno al establecimiento de modelos teóricos de la comunicación, es decir, modelos teóricos de la producción y reconocimiento de frases por parte de un hablante/oyente ideal, sino que han trabajado, fundamentalmente, en desarrollar descripciones gramaticales informatizadas, así como lenguajes formales, que puedan tener aplicaciones reales en el tratamiento informatizado de las lenguas naturales.

La realización de gramáticas y diccionarios electrónicos constituye uno de los campos en los que la investigación coordinada de la informática y la lingüística ha conseguido resultados más importantes. Dichos resultados no se han materializado únicamente en aplicaciones tecnológicas concretas dentro de lo que se ha empezado a llamar la "industria de la lengua"(2), es decir, la tecnología de la documentación automática, la generación o el análisis automático de textos, etc., sino que han puesto de manifiesto, a su vez, que las propiedades formales de las lenguas naturales son mucho más complejas de lo que se había supuesto inicialmente en la lingüística generativa. En este sentido, la constatación de la complejidad formal de las lenguas naturales ha llevado incluso a plantear una revisión del concepto de regla dentro de la lingüística transformacional (cf. Harris, 1982 y Gross, 1975).

La elaboración de un diccionario electrónico del español constituye una contribución dentro de esta nueva línea de investigación lingüística, la cual abrirá nuevas perspectivas teóricas y metodológicas en la lingüística española.

2. El diccionario electrónico consiste en una lista de entradas, que pretende constituir un léxico exhaustivo del español. Cada una de las entradas de dicha lista va acompañada de un código alfanumérico que especifica las siguientes propiedades morfológicas:

a. la categoría gramatical;

b. las propiedades de flexión de los nombres, verbos y adjetivos, que son las clases de palabras que poseen propiedades de flexión en español.

El código alfanumérico que acompaña a los verbos, los nombres y los adjetivos remite a unos ficheros de algoritmos de la flexión morfológica del español. De este modo, los programas de generación, que forman parte del diccionario electrónico, utilizan la información codificada de las propiedades morfológicas de flexión de las entradas correspondientes para generar todas las formas flexivas. La obtención de las formas flexivas mediante programas de generación y no mediante listados tiene una importancia crucial en un diccionario electrónico del español, debido a su compleja morfología flexiva; en efecto, la abundancia de formas flexivas del español hace que el número de formas de base del diccionario electrónico se multiplique por un factor 10 tras la aplicación de los programas de generación morfológica.

La selección de las formas de base de un diccionario electrónico no se puede llevar a cabo tomando --o, mucho menos aun, copiando--, las entradas de un único diccionario, aunque se trate del que tradicionalmente, se ha consierado el más "prestigiso", como el Diccionario de la Real Academia Española. En efecto, los diccionarios tradicionales están pensados para que puedan ser utilizados por lectores humanos, los cuales poseen una capacidad, o una competencia lingüística, que les permite suplir fácilmente toda aquella información que en los mencionados diccionarios o bien se encuentra de forma implícita, o bien no se especifica en absoluto. Por ello, para determinar las propiedades morfológicas de cada una de las entradas del diccionario electrónico del español, se ha tenido que añadir, en primer lugar, toda la información morfológica que no se especifica en los diccionarios tradicionales y, en segundo lugar, se ha tenido que sistematizar toda la información que en dichos diccionarios se especifica de forma más o menos dispersa.

No obstante, nuestra labor no se ha limitado únicamente a añadir, o a sistematizar, toda aquella información que como ya hemos señalado anteriormente, en los diccinarios usuales, o bien se encuentra de forma implícita, o bien no se especifica en absoluto: nuestro trabajo ha consistido, además, en el establecimiento de un nuevo criterio para determinar las entradas del diccinario electrónico. En efecto, la lexicografía tradicional mezcla criterios semánticos, etimológicos y formales para establecer los criterios que le permiten determinar una entrada desde el punto de vista lexicográfico; por el contrario, la determinación de las formas de base del diccionario electrónico del español se ha realizado utilizando un criterio de entrada basado en propiedades formales.

2. CARACTERISTICAS DE LOS PROGRAMAS DE GENERACION DE FORMAS FLEXIVAS(3)

El paquete de programas de generación de formas flexivas está formado por:

a. un programa de generación de formas nominales y adjetivas;

b. un programa de generación de formas verbales.

El programa de generación de formas nominales y adjetivas está formado por un fichero de algoritmos de la flexión nominal y adjetiva del español y por el programa de flexión propiamente dicho. Por otro lado, el paquete de programas de generación de formas verbales tiene una estructura más compleja, debido a las características de la flexión verbal del español, en la que partiendo de una única forma de base, que corresponde al infinitivo, es necesario generar un gran número de formas flexivas; el paquete de programas de generación de las formas verbales está formado dos ficheros de datos y por tres programas:

a. un fichero con los algoritmos de la flexión de los verbos regulares e irregulares en español;

b. un fichero con todas las desinencias verbales de la conjugación española, establecida y clasificada en función de los tiempos verbales a los que pertenecen;

c. un programa de pretratamiento que actúa sobre el fichero de conjugaciones y genera un nuevo fichero, que es el que realmente utiliza el programa de generación de formas verbales;

d. un programa de pretratamiento que actúa sobre el fichero de desinencias, y que, al igual que en el caso anterior, genera un nuevo fichero que es el que utiliza el programa de generación de formas verbales;

e. el programa de generación de formas verbales propiamente dicho.

El funcionamiento de los distintos programas de generación morfológica del diccionario electrónico es esencialmente el mismo. Por un lado, el programa de generación de formas nominales y adjetivas consulta el codigo alfabético que acompaña a cada una de las entradas del diccionario; en el caso de que dicho código correponda con el que se les ha asignado a los nombres y los adjetivos, es decir, N y A, el programa consulta el código numérico

de la entrada. Dicho código le remite al fichero de algoritmos de la flexión nominal y adjetiva, en el que se especifican las propiedades formales de flexión necesarias para que el programa pueda generar de forma automática todas las formas flexivas correspondientes.

El programa de flexión verbal consulta igualmente el código alfabético de las entradas del diccionario. Cuando dicho código corresponde con el que se ha asignado a los verbos, es decir, V, el programa consulta el código numérico de la entrada verbal. Dicho código le remite al fichero de algoritmos de la conjugación, en el que se dan los datos para que el programa pueda hacer el cálculo de las raíces verbales; a su vez, en dicho fichero se especifica, además, la combinación de dichas raíces con las desinencias verbales correspondientes. El programa, entonces, consulta el fichero de desinencias y genera todas las formas flexivas, correspondientes a la única forma de base que se encuentra en el diccionario electrónico, es decir, el infinitivo. En realidad, la codificación de los verbos es algo más compleja, ya que existen códigos especiales para determinar si un verbo admite una única forma para el participio, como por ejemplo, el verbo ser (cf. Figura 1), o bien varias, como por ejemplo, el verbo haber (cf. Figura 2). Asimismo, a lo largo del desarrollo de nuestro proyecto de investigación, habrá que marcar todos los verbos con respecto a sus posibilidades de combinación con uno o dos pronombres clíticos, para que los programas puedan generar, asimismo, todas las formas del imperativo, así como las del infintivo y del gerundio, que llevan uno, o dos, pronombres ortográficamente unidos al verbo, como por ejemplo, háblame, pensárselo, callándotelo, etc.

El resultado de los programas de generación morfológica los podemos observar en las figuras 1, 2 y 3. Nótese que los programas no se limitan a generar únicamente las cadenas de caracteres, correspondientes a las formas flexivas de una determinada forma de base del diccionario, sino que además especifican:

a. sus principales propiedades morfológicas;

b. las relaciones formales que mantienen con la forma de base del diccionario electrónico.

En efecto, observemos, por ejemplo, la siguiente forma verbal, generada automáticamente (cf. la Figura 2):

habremos,**********-5er.V:F1p

Además de la cadena de caracteres habremos, el programa nos proporciona una especificación codificada de la categoría gramatical a la que pertenece dicha cadena, así como una especificación de sus correspondientes propiedades morfológicas de

tiempo, modo, persona y número; en efecto, V nos indica que se trata de un verbo, F indica que se trata un tiempo futuro del modo indicativo, 1 indica que se trata de una primera persona y, finalmente, p indica que se trata de una forma verbal en plural. Los programas de generación de formas flexivas de nombres y adjetivos también dan una especificación de la categoría gramatical, así como de las correspondientes propiedades morfológicas de género y numero de las cadenas de caracteres generadas; en efecto, en (cf. Figura 3):

actrices,**********-5or.Nfp

N nos indica que se trata de un nombre, f indica que se trata de un femenino y p, que se trata de un plural.

A su vez, tanto los programas de generación de formas verbales como los de generación de formas nominales y adjetivas especifican la relación que mantiene cada una de las formas generadas con la correspondiente forma de base del diccionario electrónico. Así, en:

partiésemos,**********-6r.V:T1p

inmediantamente después de los asteriscos, que completan la línea hasta el trigésimo espacio, el programa indica, mediante un número precedido del signo "-", el número de caracteres que hay que eliminar de la cadena generada por el programa para obtener la cadena de caracteres común, tanto a la forma generada, como a la forma de base del diccionario. Así, en el ejemplo anterior, el signo "-" y el número 6, indican que hay que eliminar 6 caracteres a la cadena partiésemos, para obtener parti, que es la cadena de caracteres común a la forma generada por el programa, es decir, partiésemos, y a la forma de base del diccinario, es decir, partir. Asimismo, el programa da una indicación del carácter, o caracteres, que hay que añadir a la cadena común para obtener la forma de base. En nuestro anterior ejemplo, el programa nos indica que a la cadena que tienen en común la forma generada y el infinitivo, es decir, a parti, hay que añadirle el carácter r para obtener la forma de base partir.

Asimismo, en:

bombines,**********-4ín.Nmp

-4 indica que hay que eliminar 4 caracteres para obtener bomb, es decir, la cadena de caracteres común a la forma generada bombines y

a la forma de base bombín; asimismo, el programa indica que hay que añadir ín a la cadena común bomb para obtener la forma de base bombín.

Las indicaciones que proporciona el programa con respecto a las propiedades morfológicas de las formas generadas van a tener una importancia crucial en todas las investigaciones que utilicen el diccionario electrónico para estudiar aspectos de la estructura de la palabra en español, en relación con sus propiedades morfológicas.

3. CONCLUSION

La elaboración del diccionario electrónico del español va a permitir realizar tanto investigaciones de tipo teórico, como aplicaciones tecnológicas dentro del campo de la "industria de la lengua". Desde un punto de vista teórico, la utilización del diccionario electrónico va a permitir, por primera vez para el español, la realización de investigaciones sobre la estructura de la palabra, partiendo de la utilización de un léxico exhaustivo. La utilización de datos exhaustivos va permitir llevar a cabo investigaciones morfológicas sobre el español, partiendo de la consideración de todo el léxico y no únicamente de sectores parciales y aislados de éste, como se ha hecho hasta ahora tanto en la gramática generativa como en el estructuralismo. La investigación de la estructura del léxico en su globalidad no solo contribuirá a aumentar nuestro conocimiento sobre su estructura, sino que permitirá, además, establecer las bases de una morfología científica, sólidamente basada en un conocimiento empírico del léxico. Asimismo, en la medida en la que el diccionario electrónico pretende ser un reflejo exhaustivo del léxico español, puede constituir una base de datos fiable con un porcentaje de error nulo en sus aplicaciones tecnológicas en el campo de la "industria de la lengua".

```
ser,************************.V:W00
siendo,*********************-5er.V:G00
sido,***********************-3er.V:Kms
soy,************************-2er.V:P1s
eres,***********************-4ser.V:P2s
es,*************************-2ser.V:P3s
somos,**********************-4er.V:P1p
sois,***********************-3er.V:P2p
son,************************-2er.V:P3p
era,************************-3ser.V:I1s:I3s
eras,***********************-4ser.V:I2s
éramos,*********************-6ser.V:I1p
erais,**********************-5ser.V:I2p
eran,***********************-4ser.V:I3p
fui,************************-3ser.V:J1s
fuiste,*********************-6ser.V:J2s
fue,************************-3ser.V:J3s
fuimos,*********************-6ser.V:J1p
fuisteis,*******************-8ser.V:J2p
fueron,*********************-6ser.V:J3p
seré,***********************-1.V:F1s
serás,**********************-2.V:F2s
será,***********************-1.V:F3s
seremos,********************-4.V:F1p
seréis,*********************-3.V:F2p
serán,**********************-2.V:F3p
sea,************************-1r.V:S1s:S3s
seas,***********************-2r.V:S2s
seamos,*********************-4r.V:S1p
seáis,**********************-3r.V:S2p
sean,***********************-2r.V:S3p
fuera,**********************-5ser.V:T1s:T3s
fueras,*********************-6ser.V:T2s
fuéramos,*******************-8ser.V:T1p
fuerais,********************-7ser.V:T2p
fueran,*********************-6ser.V:T3p
fuese,**********************-5ser.V:T1s:T3s
fueses,*********************-6ser.V:T2s
fuésemos,*******************-8ser.V:T1p
fueseis,********************-7ser.V:T2p
fuesen,*********************-6ser.V:T1s
sé,*************************-1er.V:Y2p
sed,************************-1r.V:Y2p
sería,**********************-2.V:C1s:C3s
serías,*********************-3.V:C2s
seríamos,*******************-5.V:C1p
seríais,********************-4.V:C2p
serían,*********************-3.V:C3p
```

Figura 1. Conjugación automática del verbo ser

```
haber,**********************.V:W00
habiendo,********************-5er.V:G00
habido,**********************-3er.V:Kms
habida,**********************-3er.V:Kfs
habidos,*********************-4er.V:Kmp
habidas,*********************-4er.V:Kfp
he,**************************-1aber.V:P1s:Y2p
has,*************************-1ber.V:P2s
ha,**************************-0ber.V:P3s
hemos,***********************-4aber.V:P1p
habéis,**********************-3er.V:P2p
han,*************************-1ber.V:P3p
había,***********************-2er.V:I1s:I3s
habías,**********************-3er.V:I2s
habíamos,********************-5er.V:I1p
habíais,*********************-4er.V:I2p
habían,**********************-3er.V:I3p
hube,************************-3aber.V:J1s
hubiste,*********************-6aber.V:J2s
hubo,************************-3aber.V:J3s
hubimos,*********************-6aber.V:J1p
hubisteis,*******************-8aber.V:J2p
hubieron,********************-7aber.V:J3p
habré,***********************-2er.V:F1s
habrás,**********************-3er.V:F2s
habrá,***********************-2er.V:F3s
habremos,********************-5er.V:F1p
habréis,*********************-4er.V:F2p
habrán,**********************-3er.V:F3p
haya,************************-2ber.V:S1s:S3s
hayas,***********************-3ber.V:S2s
hayamos,*********************-5ber.V:S1p
hayáis,**********************-4ber.V:S2p
hayan,***********************-3ber.V:S3p
hubiera,*********************-6aber.V:T1s:T3s
hubieras,********************-7aber.V:T2s
hubiéramos,******************-9aber.V:T1p
hubierais,*******************-8aber.V:T2p
hubieran,********************-7aber.V:T3p
hubiese,*********************-6aber.V:T1s:T3s
hubieses,********************-7aber.V:T2s
hubiésemos,******************-9aber.V:T1p
hubieseis,*******************-8aber.V:T2p
hubiesen,********************-7aber.V:T1s
habed,***********************-1r.V:Y2p
habría,**********************-3er.V:C1s:C3s
habrías,*********************-4er.V:C2s
habríamos,*******************-6er.V:C1p
habríais,********************-5er.V:C2p
habrían,*********************-4er.V:C3p
```

Figura 2. Conjugación automática del verbo haber

```
andaluz,**********.Nms
andaluza,**********-1.Nfs
andaluces,**********-3z.Nmp
andaluzas,**********-2.Nfp
andaluz,**********.Ams
andaluza,**********-1.Afs
andaluces,**********-3z.Amp
andaluzas,**********-2.Afp
monje,**********.Nms
monja,**********-1e.Nfs
monjes,**********-1.Nmp
monjas,**********-2e.Nfp
abad,**********.Nms
abadesa,**********-3.Nfs
abades,**********-2.Nmp
abadesas,**********-4.Nfp
alemán,**********.Nms
alemana,**********-3án.Nfs
alemanes,**********-4án.Nmp
alemanas,**********-4án.Nfp
alemán,**********.Ams
alemana,**********-3án.Afs
alemanes,**********-4án.Amp
alemanas,**********-4án.Afp
zar,**********.Nms
zarina,**********-3.Nfs
zares,**********-2.Nmp
zarinas,**********-4.Nfp
rey,**********.Nms
reina,**********-3y.Nfs
reyes,**********-2.Nmp
reinas,**********-4y.Nfp
bailarín,**********.Nms
bailarina,**********-3ín.Nfs
bailarines,**********-4ín.Nmp
bailarinas,**********-4ín.Nfp
actor,**********.Nms
actriz,**********-3or.Nfs
actores,**********-2.Nmp
actrices,**********-5or.Nfp
emperador,**********.Nms
emperatriz,**********-4dor.Nfs
emperadores,**********-2.Nmp
emperatrices,**********-6dor.Nfp
```

Figura 3. Flexión automática de algunos nombres y adjetivos

NOTAS

1. Queremos dar las gracias a D.Bernabé, J.B.Carreño, Ll.Ferrer, S.Galera, I.Nuñez, T.Peral, M.Rodriguez y J.Torrecilla, así como a los restantes miembros del Centro de Cálculo de la Universidad Autónoma de Barcelona por toda la ayuda que nos han ofrecido. También queremos expresar nuestro agradecimiento a Annibale Elia, así como a todos los miembros de su equipo de investigación del Istituto di Linguistica de la Università di Salerno (Italia) por su importante colaboración. Queremos expresar, asimismo, nuestro agradecimiento a M.Geffriaud, A.Guillet, G.Klarsfeld, C.Leclère, M.Piot, Ph.Vasseux, así como a todos los restantes miembros del Laboratoire d'Automatique Documentaire et Linguistique de la Université Paris 7, por toda la ayuda científica y técnica que nos han ofrecido. La ayuda de Blandine Courtois ha sido crucial. La realización de este proyecto no hubiese sido posible sin la ayuda de Maurice Gross.

   Este proyecto se está realizando con una subvención de la CAICYT (PB85-0371-C02-01) del Ministerio de Educación y Ciencia y una ayuda de Acción Integrada Hispano-Francesa (9/322).

2. Cf. Maurice Gross. 1986. "Une mise à plat du français écrit", Les industries de la langue, Université Paris 8, pp.51-55.

3. Los programas de generación de formas verbales han sido escritos por Blandine Courtois (cf. Programas utilizados en las Referencias); dichos programas han sido transferidos al Departamento de Filología Española de la Universidad Autónoma de Barcelona por el Laboratoire d'Automatique Documentaire et Linguistique de la Université Paris 7, dentro del acuerdo oficial de cooperación científica que existe entre ambas instituciones.

REFERENCIAS

1. PROGRAMAS UTILIZADOS

1. Programa de pretratamiento del fichero de desinencias verbales, Blandine Courtois, Laboratoire d'Automatique Documentaire et Linguistique (CNRS), Université Paris 7

2. Programa de pretratamiento del fichero de conjugaciones, Blandine Courtois, Laboratoire d'Automatique Documentaire et Linguistique (CNRS), Université Paris 7

3. Programa de generación de formas verbales, Blandine Courtois, Laboratoire d'Automatique Documentaire et Linguistique (CNRS), Université Paris 7

4. Programa de generación de formas nominales y adjetivas, Blandine Courtois, Laboratoire d'Automatique Documentaire et Linguistique (CNRS), Université Paris 7

2. REFERENCIAS BIBLIOGRAFICAS

* COURTOIS, Blandine. 1984. DELAS: Dictionnaire Électronique du LADL. Mots simples, Rapport Technique du LADL No.12.

* CUERVO, Rufino J. 1886 y 1893. Diccionario de construcción y régimen de la lengua castellana, 2 vols. Bogotá: Instituto Caro y Cuervo, 1953 y 1954.

* DANLOS, Laurence. 1987. The Linguistic Basis of Text Generation. Cambridge: Cambridge University Press.

* ELIA, Annibale. 1986. Le dictionnaire électronique de l'italien, Istituto di Linguistica, Università di Salerno.

* GROSS, Maurice. 1975. Méthodes en syntaxe. Paris: Hermann.

* GROSS, Maurice. 1968-1987. Grammaire française, 3 vols; Syntaxe du verbe, vol.I; Syntaxe du nom, vol.II; Syntaxe de l'adverbe, vol.III. Paris: Cantilène.

* MOLINER, María. 1975. Diccionario de uso del español, 2 vols. Madrid: Gredos.

* NEBRIJA, Antonio de. 1503.De ui ac potestate litterarum, trad. y ed. A.Quilis y P.Usábel. Madrid: SGEL, 1987.

* SOPEÑA, Luis. 1987. Diccionario del castellano en ordenador para la composición y verificación de textos, Centro de investigación UAM-IBM, Madrid.

* SUBIRATS RÜGGEBERG, C. 1987. Sentential Complementation in Spanish. Amsterdam/Philadelphia: John Benjamins.

* SUBIRATS RÜGGEBERG, C. (en prensa). El diccionario electrónico del español, Actas del III Coloquio de la Sociedad Española de Procesamiento del Lenguaje Natural, Madrid.

On the Connections Between Relational Tableaux and Clause Form

Pedro Manoel Silveira

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica

ABSTRACT

The theory of tableaux for relational databases is applied to the test of containment among relational expressions. In this paper we analyse the connections between tableaux and clause form representation. We show that results from the theory of tableaux are also obtainable with clause form.

1. Introduction

The test of equivalence of queries is a basic step towards the obtention of methods for database query optimization. The main results in this topic seem to be the ones published in [1,7,9]. where the authors introduce and further extend the concept of relational tableaux. These provide a tabular set description of queries and methods have been developed to carry out containment tests among tableaux instances. Despite their obvious utility tableaux, as they stand, do not support the complete range of relational expressions. The main problems remain around the use of the difference operator and some other minor points [4].

This paper deals with the connections between relational tableaux and some techniques used in the manipulation of logical formulae. The relevance of this work is the exploration of new approaches for the complete solution of the problem of query containment for unrestricted relational expressions.

This paper is organized as follows. Section 2 and 3 contain brief reviews of relational tableaux and clause form representation. Section 4 presents concepts and definitions covering the subject and formally introduces the central arguments of our work. Section 5 discusses these results and hints further research on the subject.

A	B	C
x	y	

(xy | for some z, xyz ∈ I)

A	B	C
	y	z
x	y	z

(yz | for some x, xyz ∈ I)

A	B	C
x	y	z
x	y	v_a
v_b	y	z

(xyz | for some v_a, v_b, xyv_a ∈ I & $v_b$yz ∈ I)

Figure 1. Examples of Relational Tableaux

2. Relational Tableaux

Tableaux were introduced by Aho, Sagiv and Ullman [1] and provide a tabular set description. A tableau is a matrix, where the first row is called summary and the remaining rows are called rows. The summary represents the target list of a set expression and the rows represent the tuples required to be in the universe I, supposing there is a relation R on the set of all attributes whose instance is I. The columns of a tableau correspond to the attributes of R in a fixed order. Figure 1 shows examples of tableaux, with their corresponding set expressions.

The first two examples correspond to relation schemata AB and BC respectively, and the third to the natural join of AB and BC. In the example, x, y, z are said to be distinguished variables, because they appear in the target list of the corresponding set expressions, while v_a and v_b are said to be nondistinguished variables, because they appear only on the right side of the bar. Constants and blank symbols can also appear in a tableau.

T(I) is a relation defined on the atributes whose columns in the summary are non-blanks, and is defined as

$$T(I) = \{ \vartheta(w_0) \mid \text{for some valuation } \vartheta,\ \vartheta(w_i) \in I,\ 1 \le i \le n \}$$

A valuation ϑ associates with each symbol of T a constant c. For a row w_i, $\vartheta(w_i)$ is the tuple obtained by substituting

i) $\vartheta(v)$ for every variable v appearing in w_i;

ii) $\psi(c)=c$ for every constant c appearing in w_1.

3. Clause Form

Clause form [63] is a normalized representation of logical formulae. In general, any Predicate Calculus formula can be represented by a conjunction of clauses, where each clause contains only the disjunction of literals. The following transformations are necessary to translate a formula to its clause form representation.

a) the elimination of implication signs, replacing

$$\phi_1 \rightarrow \phi_2 \quad \text{by} \quad \sim\phi_1 \mid \phi_2$$

b) the reduction of the scopes of the negation symbols, replacing

$$\begin{array}{rcl} \sim\sim\phi & \text{by} & \phi \\ \sim(\phi_1 \,\&\, \phi_2) & \text{by} & \sim\phi_1 \mid \sim\phi_2 \\ \sim(\phi_1 \mid \phi_2) & \text{by} & \sim\phi_1 \,\&\, \sim\phi_2 \\ \sim\forall x\phi & \text{by} & \exists x\sim\phi \\ \sim\exists x\phi & \text{by} & \forall x\sim\phi \end{array}$$

c) the renaming of variables, such that no two quantifiers apply to variables with identical names

d) the elimination of existential quantifiers, through the introduction of Skolem functions. The idea is to replace the occurrence of existentially quantified variables by functions that represent elements in the domain of the original variables. These functions take as arguments the universally quantified variables whose scopes reach the existential quantifiers being removed.

e) the conversion to prenex normal form. A formula is in prenex normal form if and only if it has the form of a list of quantifiers followed by a quantifier free formula (called the matrix). At this stage, there can be only universal quantifiers and they can be simply moved to the front of the formula, following

$$(\forall x\ \phi_1) \,\&\, \phi_2 \;=\; \forall x(\phi_1 \,\&\, \phi_2)$$

and

$$(\forall x\ \phi_1) \mid \phi_2 \;=\; \forall x(\phi_1 \mid \phi_2)$$

if x is not free in ϕ_2.

f) the conversion of the matrix to conjunctive normal form. A formula is in conjunctive normal form when it is formed by a conjunction of clauses. This can be done by replacing

$$\phi_1 \mid (\phi_2 \,\&\, \phi_3) \quad \text{by} \quad (\phi_1 \mid \phi_2) \,\&\, (\phi_1 \mid \phi_3)$$

g) the elimination of universal quantifiers. All variables remaining at this stage are universally quantified. Assuming that the variables in the matrix remain so, we can drop the universal

quantifiers altogether.

h) the elimination of the logical AND connectives. This way we are left with a finite set of clauses, which are disjunctions in the matrix resulting from step g.

A method for translating a query to clause form is now shown. Let Q be a query of the form

$$t_1,\ldots,t_n \mid v_1 \in D_1, \ldots, v_k \in D_k \mid \phi(v_1, \ldots, v_n)$$

Taking Q as a hypothetical relation containing all and only the tuples forming the answer to a query Q in the format above, we can say that

$$\forall v_1 \in D_1 \ldots \forall v_k \in D_k \quad \phi(v_1, \ldots, v_n) \rightarrow Q(t_1,\ldots,t_n) \qquad (1)$$

Translating formula 1 into clause form, as indicated above, we have a set of clauses of the form

$$Q(t_1,\ldots,t_n) \mid \Psi_i$$

where Ψ_i is a possibly empty disjunction of literals, t_i is a term and all variables in each clause are universally quantified over their respective domains.

4. Clause Form and Relational Tableaux

We now show the connection between clause form representation and the theory of tableaux for relational databases.

To represent tableaux we use a very simple method. Taking Q as the virtual relation containing all and only the tuples in the answer for a query Q, we can write a clause

$$Q(\text{summary}) \mid \sim I(\text{row}_1) \mid \ldots \mid \sim I(\text{row}_n)$$

for a tableau with summary and n rows. This can be verifiable from the definitions for tableaux. Notice that each term in the literal in Q must have a specified domain. In a tableau, the elements of the summary have their domains positionally determined. In the translation to clause form, it is necessary to indicate the attribute in R corresponding to each term in the summary. So, for the third tableau in Figure 1 we would have the clause

$$Q(x,y,z) \mid \sim I(x,y,v_1) \mid \sim I(v_2,y,z)$$

Before proceeding, we must introduce some concepts for tableaux and clause manipulation, borrowing material from [1,2,6].

Definition. Let T_1 and T_2 be tableaux, whose sets of symbols are E_1, E_2 respectively. A homomorphism is a mapping $m: E_1 \rightarrow E_2$, such that

a) if c is a constant, $m(c)=c$;
b) if v is a distinguished variable, then $m(v)$ is either a distinguished variable or a constant;
c) if w is any row of T_a then $m(w)$ is a row of T_b.

Definition. Let T_a and T_b be tableaux and let m be a mapping from the rows of T_a to the rows of T_b. m is a containment mapping if:

a) if T_a has a distinguished variable in column A of row i then T_b has a distinguished variable or a constant in column A of row $m(row_i)$ of T_b;
b) if T_a has a constant c in column A of row i then T_b has a constant c in row $m(row_i)$;
c) if rows i and j of T_a have the same nondistinguished variable in column A, then rows $m(row_i)$ and $m(row_j)$ of T_b have the same symbol on that column.

Definition. A substitution is a possibly empty set of the form $(t_1/v_1, \ldots, t_n/v_n)$, where v_i is a variable and t_i is a term, different from v_i, such that no two variables v_i, v_j, $i=j$, are the same.

Definition. Let G be a substitution and let E be an expression. The instantiation of E (or instance of E) by G is denoted $G.E$ and is the expression E where each occurrence of the symbol v_i has been replaced by its corresponding t_i in G.

Definition. A clause C_a subsumes another clause C_b if and only if there is a substitution θ such that $\theta.C_a \subseteq C_b$. C_b is said to be subsumed by C_a.

Theorem 1. $T_b \subseteq T_a$ (T_b is contained in T_a) if and only if they define the same target relation and have a containment mapping m from T_a to T_b. Proved in [1].

The definitions above introduce some necessary concepts we use ahead. The central idea here is to find a containment mapping between the two tableaux in order to verify their containment. Let T_a and T_b be tableaux. We must prove that $T_b \subseteq T_a$ with m if and only if there exists a substitution G such that $G.C_a \subseteq C_b$, where C_a and C_b are the clauses corresponding to T_a and T_b respectivelly. Recall that if $\theta.C_a \subseteq C_b$, then we say that C_a subsumes C_b. In other words, if the clauses C_a and C_b obtained from tableaux T_a, T_b respectivelly, admit a substitution G such that $G.C_a \subseteq C_b$ or $G.C_b \subseteq C_a$, then we can infer the existence of a containment mapping between the corresponding tableaux. That is, we can test the containment of tableaux using their corresponding clause form representation. In doing this, we are in fact testing the containment of two relational expressions. The equivalence of tableaux is decided from their containment. $T_a \equiv T_b$ if and only if $T_a \subseteq T_b$ and $T_b \subseteq T_a$. Similarly, $T_a \not\equiv T_b$ if and

only if C_s subsumes C_m and C_m subsumes C_s.

Lemma. For each homomorphism $m: E_s \rightarrow E_m$ there exists an equivalent substitution G, such that $G.E_s \equiv E_m$.

PROOF. According with the symbols in E_s, we can always build G as follows:

a) if c is a constant, then $m(c)=c$, and no substitution for c appears in G;
b) if v_i is a distinguished variable and $m(v_i)=v_j$, the pair v_j/v_i is in G;
c) if v_i is a distinguished variable and $m(v_i)=t$, where t can be distinguished, nondistinguished or a constant, the pair t/v_i is in G.

Theorem 2. Let T_s and T_m be tableaux whose corresponding clause forms are C_s and C_m respectively. $T_m \subset T_s$ if and only if there exists G such that C_s subsumes C_m, viz. $G.C_s \subseteq C_m$.

PROOF: If direction.
Suppose G exists. Then $T_m \subset T_s$.
a) if w is a row of T_s and $m(w)$ is in T_m, then if L is a literal in C_s necessarily $G.L$ is a literal in C_m;
b) if T_s and T_m have the same target relation schema, then the literal in Q of C_s maps to the literal in Q of C_m through G.

Only if.
Suppose G does not exist. There can be three reasons for that:
a) the literals in Q in clauses C_s and C_m have different numbers of components, or their corresponding terms have different domains. In this case, T_s cannot contain T_m;
b) the literals in Q have constants c_s and c_m, $c_s \neq c_m$, in corresponding columns. In this case, T_s cannot contain T_m, either because they contain different summaries or because their summaries contain distinguished variables that have been mapped to different terms;
c) for some literal L in C_s, there is a constant c as component i, such that for any literal in I in C_m, $G.$component i is a constant different from c. In this case, T_s cannot contain T_m because if w is the row of T_s corresponding to L, then $m(w)$ is not in T_m.

Another major benefit from the theory of tableaux is the row elimination rule, i. e., the simplification of queries. This comes as a corollary to Theorem 1, and is proved in [13].

Corollary. Let T be a tableau, w and x rows of T. If in whatever column w and x disagree, and w has a nondistinguished variable that appears nowhere else in T, then row w can be eliminated, and the resulting tableau is

equivalent to T.

We state without proof its equivalent for clause form.

Theorem 3. Let T be a tableau, **w** and **x** rows of T. **w** can be eliminated by the rule above if and only if clause **C**, corresponding to T can be simplified by the removal of its literal corresponding to row **w**.

The proof is based on the fact that the variable in **C** corresponding to the nondistinguished variable in **w** that disagrees with some component of **x**, can be replaced by that component, resulting in a clause **C'**, such that **C'** subsumes **C**.

5. Conclusions

In the sections above we showed how some results from the theory of tableaux can be obtained in a framework of clause form representation for database queries. This section discusses some of the implications of such and directions for further research.

The use of clause form is a useful representation due to the possibility of using other methods for query optimization. Semantic query optimization [3,5], for example, is well suited in this framework and combines logical inference with the treatment of clauses.

An issue not mentioned in the text is the use of functional dependencies. Basically, if **X -> Y** is a functional dependency and if rows i and j of tableau **T** have identical symbols in the columns corresponding to the attributes in **X**, then the symbols corresponding to columns **Y** should be made identical for both rows. This possibility is covered in clause form by the acceptance of function symbols in a better structured way. That is, functions can be directly represented as terms and substitution can take them into account.

Sagiv & Yannakakis [7] extended the theory of tableaux for the union operation and to a restricted family of relational expressions containing the set difference operator. For monotonic relational expressions, viz, the ones where only the operators select, projection, join and union are used, the authors show that the equivalence and containment of tableaux can be characterized in terms of containment of single tableaux. That permits us to infer that the proofs presented above enable the assumption that monotonic expressions are covered by clause form. The difficulty of tableaux in dealing with difference expressions can be traced to its inability to handle function symbols. In clause form however, this is solved by the introduction of Skolem functions for existential quantifiers and the handling of function symbols. In any case, Sagiv & Yannakakis express differences in terms of tableaux expressions, consisting of tableaux as operands and union, intersection and difference as operators.

In [8], we investigate the generalization of clause form representation to the complete set of relational operations, in order to analyse the connections between clause form representation and the work cited above.

6. References

1. Aho, A., Sagiv, Y. and Ullman, J. Equivalences Among Relational Expressions, SIAM J. Comput. 8,2(May 1979), 218-246.

2. Chang, C. and Lee, R., Symbolic Logic and Mechanical Theorem Proving, Academic Press, 1973.

3. Hammer, M. and Zdonik-Jr, S. B., Knowledge-Base Query Processing, Proc. of the 6th Int. Conf. VLDB, 1980.

4. Jarke, M. and Koch, J., Query Optimization in Database Systems, Computing Surveys 16,2(June 1984).

5. King, J., QUIST: A System for Semantic Query Optimization, Proc. of the 7th Int. Conf. VLDB, 1981.

6. Nilsson, N. J., Problem Solving Methods in Artificial Intelligence, Mc-Graw Hill, 1971.

7. Sagiv, Y. and Yannakakis, M. Equivalences Among Relational Expressions with The Union and Difference Operators, Journal of the ACM 27,4(October 1980), 633-655.

8. Silveira, P. M. Equivalences Among Relational Expressions with Clause Manipulation, to appear as Technical Report NCE/UFRJ.

9. Ullman, J. D., Principles of Database Systems 2nd ed., Computer Science Press, 1982.

ON A MIN-MAX CONJECTURE FOR REDUCIBLE DIGRAPHS

Jayme Luiz Szwarcfiter
Universidade Federal do Rio de Janeiro
Núcleo Comp. Eletr. e Inst. Matemática
Caixa Postal 2324
20.001 - Rio de Janeiro, RJ
BRASIL

ABSTRACT

A. Frank and A. Gyárfás (1976) have conjectured that in a reducible digraph D the maximum number of edge disjoint cycles equals the minimum number of edges intersecting all cycles of D. We prove this conjecture in the special case when D has at most two distinct dominators. The proof leads to a polynomial time algorithm for finding both the maximum set of cycles and minimum set of edges, in the considered case.

RESUMO

A. Frank e A. Gyárfás (1976) conjecturaram que em um dígrafo redutível D o número máximo de ciclos disjuntos em arestas é igual ao número mínimo de arestas que interceptam todos os ciclos de D. Provamos essa conjectura no caso especial em que D possui no máximo dois denominadores distintos. A prova conduz a um algoritmo polinomial para encontrar tanto o conjunto máximo de ciclos quanto o conjunto mínimo de arestas, no caso considerado.

1. Introduction

A conjecture by A.Frank and A.Gyárfás [2] states that the maximum number of edge disjoint cycles of a reducible digraph D equals the minimum number of edges whose removal turns D acyclic. In the present paper we prove a special case of this conjecture. The proof is constructive and leads to a polynomial time algorithm for finding such a maximum set of cycles and minimum set of edges, in the considered case.

A flow digraph is a digraph D together with a vertex $s \in V(D)$, called root, such that every vertex of D is reachable from s. In particular, if every path from s to $v \in V(D)$ contains $w \in V(D)$ then w dominates v. A (fully) reducible digraph is a flow digraph D such that each cycle C of D contains some vertex w which dominates all the vertices of C. We call w a dominator of both C and D. See [5,7,9].

Let D be a general digraph. Denote by

α_V = set of vertex disjoint cycles of D

α_E = set of edge disjoint cycles of D

β_V = set of vertices intersecting all cycles of D

β_E = set of edges intersecting all cycles of D

Clearly, $\max|\alpha_V| \leq \min|\beta_V|$ and $\max|\alpha_E| \leq \min|\beta_E|$.

β_V and β_E are also known as feedback vertex and edge sets, respectively. Recall that the problems of finding the minimum cardinality sets α_V and α_E are both NP-hard [3,6].

Theorem 1 (Frank and Gyárfás [2]): If D is reducible then $\max|\alpha_V(D)| = \min|\beta_V(D)|$.

It follows that a minimum feedback vertex set of a reducible digraph can be found in polynomial time [2,4,8].

Conjecture [2]: If D is reducible then $\max|\alpha_E(D)| = \min|\beta_E(D)|$.

We prove this conjecture in the case when D has at most two distinct dominators.

2. The Proof

Throughout this section, D will always denote a reducible digraph.

Let $C \equiv v_1,\ldots,v_k,v_1$, $k>1$, be a cycle of D and v_1 the dominator of C. Then edge (v_k,v_1) is called a back edge of D.

Lemma 1: Each cycle of D contains exactly one back edge.

Proof: Let $v_1,\ldots,v_k,v_1$ be a cycle of D and v_1 its dominator. Then (v_k, v_1) is a back edge. Suppose that C contains another back edge (v_i,v_{i+1}), $i\neq k$. In this case, v_{i+1} is the dominator of another cycle C' which contains v_i. Let P be a path from s to v_1 followed by $v_1,\ldots,v_i$. Then P meets C' in a vertex which is not its dominator, a contradiction. That is, C has exactly one back edge.

Let $\{r_1,\ldots,r_m\}$ be the set of dominators of D. Denote by D* the network obtained by the following construction:

1. Remove all back edges of D. Let D_A be the resulting acyclic digraph.

2. Assign a distinct positive label $x(r_i)$ to each dominator r_i, such that if r_i reaches r_j in D_A then $x(r_i) < x(r_j)$, $1 \leq i,j \leq m$ and $i \neq j$.

3. For each back edge (v,r_j) of D, include a new vertex w and an edge (v,w). Assign to w the negative label $-x(r_j)$.

4. Include the ordered sets

$$S=\{s_1,\ldots,s_m\} \quad \text{and} \quad T=\{t_1,\ldots,t_m\}$$

of new vertices. S and T are the sources and sinks of D*, respectively. For $1 \leq j \leq m$, include the edge (s_j,r_j) and an edge to t_j from every vertex with label $-x(r_j)$. Assign to s_j and t_j the labels $x(r_j)$ and $-x(r_j)$, respectively.

5. Assign capacity 1 to each edge of D_A and infinite to those leaving S and entering T.

The sets $S=\{s_i\}$ and $T=\{t_i\}$ are in <u>normal order</u> if $x(s_i)<x(s_{i+1})$ and $x(t_i)>x(t_{i+1})$, $1 \leq i \leq m$, respectively.

<u>Lemma 2</u>: Let $S=\{s_1,\ldots,s_m\}$ and $T=\{t_1,\ldots,t_m\}$ be the sources and sinks of D*, respectively in normal order. Then there is a one-to-one correspondence between cycles of D and s_j-t_j paths of D*.

<u>Proof</u>: Let $v_1,\ldots,v_k,v_1$ be a cycle C of D and v_1 its dominator. Then (v_k,v_1) is a back edge and there exists $w \in V(D^*)-V(D)$ such that $(v_k,w) \in E(D^*)$ and $x(w)=-x(v_1)$. Consequently, for some j, $1 \leq j \leq m$, $(s_j,v_1),(w,t_j) \in E(D^*)$ and therefore $s_j,v_1,\ldots,v_k,w,t_j$ is a path in D*. The converse is similar.

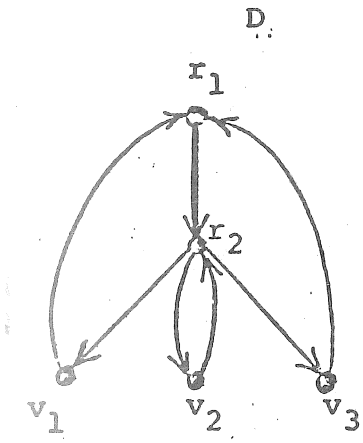
Lemma 3: Let $\{r_1, r_2\}$, $S=\{s_1, s_2\}$ and $T=\{t_1, t_2\}$ be the dominators, sources and sinks of D^* in normal order, respectively. Denote by f a S-T flow in D^* having value n and such that $f(s_1) \geq f(t_1)$. Then D has at least n edge disjoint cycles.

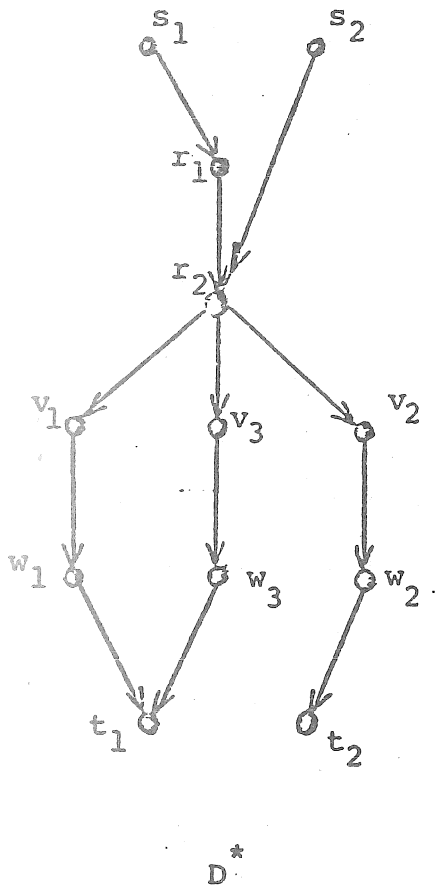
Proof: Since f has value n, D^* contains a set P, $|P|=n$, of S-T edge disjoint paths. Divide P into four subsets P_1, P_2, P_3 and P_4, consisting respectively of s_1-t_1, s_2-t_2, s_1-t_2 and s_2-t_1 paths. Clearly, $|P_3| = f(s_1) - |P_1|$ and $|P_4| = f(t_1) - |P_1|$. Hence $|P_3| \geq |P_4|$. We obtain the required cycles as follows. Each path of P_1 or P_2 corresponds to a cycle of D, according to lemma 2. Since D is reducible, each s_1-t_2 path contains r_2. Consequently, the union of a pair of cycles, one of P_3 and the other of P_4, contains two disjoint paths, of types s_1-t_1 and s_2-t_2, respectively, that is, two new cycles of D. In addition, the $|P_3|-|P_4|$ remaining s_1-t_2 paths can be transformed into an equal number of s_2-t_2 paths, by disregarding the s_1-r_2 subpaths and adding edges (s_2, r_2). A total of n edge disjoint cycles of D has been obtained.

If $k>0$ is an integer denote by kD* the network obtained from D as follows:

1. Construct D^*. Let $D_1^*, \ldots, D_k^*$ be k identical copies of D^*, with $S_i=\{s_{i1}, \ldots, s_{im}\}$ and $T_i=\{t_i, \ldots, t_{im}\}$, respectively the sets of sources and sinks of D_i^*, in normal order. The vertices of S_1 and T_k are the sources and sinks of kD*, respectively.

2. For $1 \leq i \leq k$ and $1 \leq j \leq m$ include an edge $(t_{ij}, s_{i+1,j})$ and assign to it infinite capacity.





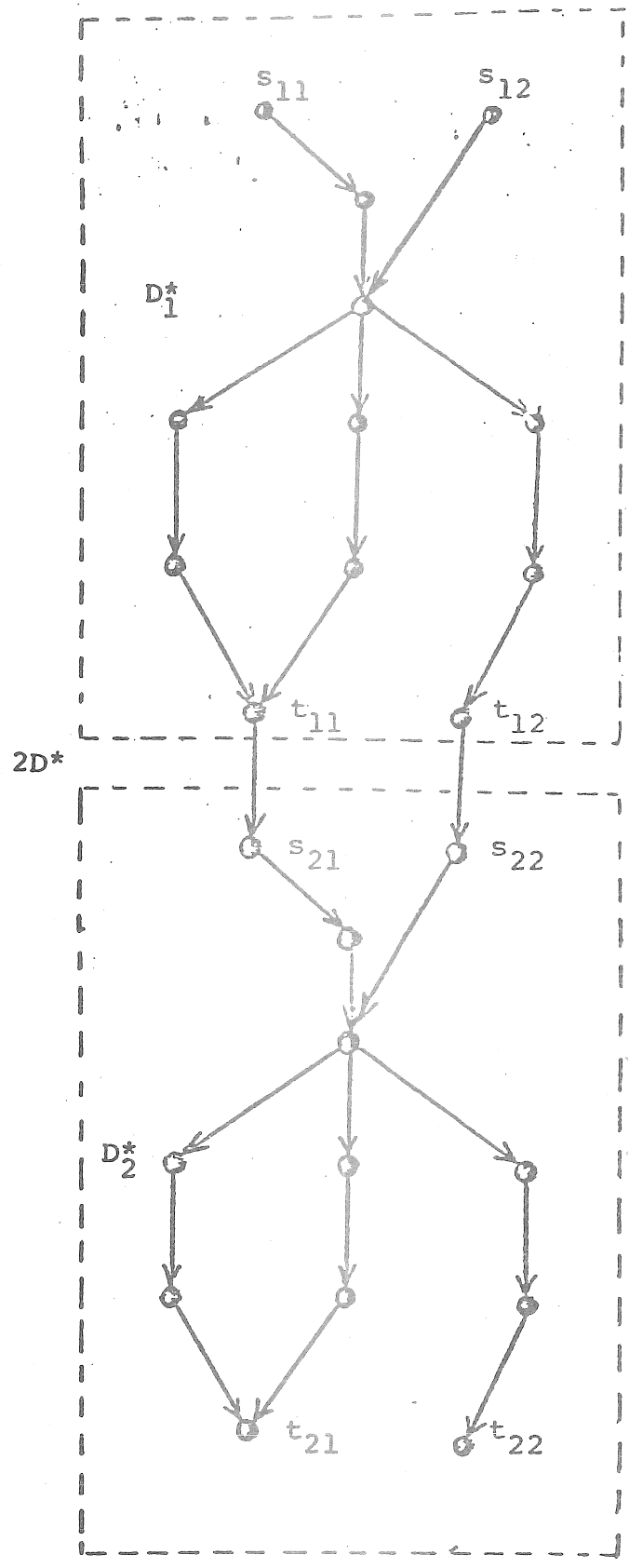


Figure 1 : A reducible digraph D and its associated networks.

Lemma 4: If kD^* has a cut X of capacity $n<\infty$ then D has a feedback edge set β_E, such that $|\beta_E| \leq n$.

Proof: Let $Q=\{e \in E(D) \mid e \in X\}$. Then $|Q| \leq n$. Next, we show that Q is a feedback edge set. Suppose it is not. Then by lemma 2, D^* has a path p of the type s_j-t_j containing no edge of Q. By repeating p in each copy D_i^* of kD^* we obtain a S_1-T_k path in kD^* with no edge of X, a contradiction▲

Theorem 2: If $m \leq 2$, $\max|\alpha_E(D)| = \min|\beta_E(D)|$.

Proof: If $m=1$ the theorem follows from lemma 2 and the max-flow min-cut theorem [1] applied to D^*. When $m=2$, construct kD^*, $k=|E(D)|$. Let $S_1=\{s_{11}, s_{12}\}$ and $T_k=\{t_{k1}, t_{k2}\}$ be the sets of sources and sinks of kD^*, respectively in normal order, and f a maximum S_1-T_k flow in kD^*, having value n. Suppose, initially, $f(s_{i1}) < f(t_{i1})$, for all $1 \leq i \leq k$. Since $f(s_{i1}) + f(s_{i2}) = f(t_{i1}) + f(t_{i2}) = n$, it follows that $f(s_{i2}) > f(t_{i2})$, $1 \leq i \leq k$. Because $f(t_{i2}) = f(s_{i+1,2})$, $1 \leq i < k$, we conclude that $f(s_{12}) > f(s_{22}) > \ldots > f(s_{k2})$. However, the latter inequality can not occur because $f(s_{12}) < |E(D)|$, $k=|E(D)|$ and all flow values $f(s_{i2})$ are non negative integers. Consequently, there exists some j, $1 \leq j \leq k$, such that $f(s_{j1}) \geq f(t_{j1})$. Applying lemma 3 to D_j^*, we conclude that D contains at least n edge disjoint cycles, that is, $\max|\alpha_E(D)| \geq n$. By the max-flow min-cut theorem, kD^* has a cut of capacity n and using lemma 4 it follows that D contains a feedback edge set of size $\leq n$, that is, $\min|\beta_E(D)| \leq n$. Hence $\max|\alpha_E(D)| \geq \min|\beta_E(D)|$. Since $\max|\alpha_E(D)| \leq \min|\beta_E(D)|$ the equality follows▲

3. The Algorithms

The algorithms follow from the proof. Given the reducible digraph D, construct the network kD^*, $k=|E(D)|$. Then find the minimum cut X of kD^*. The edges of D which form X constitute the minimum cardinality feedback edge set β_E. For finding a maximum set of disjoint cycles α_E, let f be the maximum S-T flow in $|E(D)|D^*$ Next, identify the copy D_j^* in $|E(D)|D^*$ such that $f(s_{j1}) \geq f(t_{j1})$. Then use the construction of lemma 4 which transforms f into the desired α_E.

Both algorithms, for finding a minimum feedback edge set and a maximum set of edge disjoint cycles, have the same complexity as finding a maximum S-T flow in the network $|E(D)|D^*$, that is, polynomial in $|V(D)|$.

REFERENCES

[1] L.R.Ford and D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, 1962.

[2] A.Frank and A.Gyárfás, Colloques Internationaux C.N.R.S. Problèmes Combinatoires et Théorie des Graphes (260), 157-158, 1976.

[3] M.Garey and D.S.Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H.Freeman, San Francisco, 1979.

[4] A.Gyárfás, Blocking the Loops of a Flow-Graph, Zeits. Angew. Math. Mech. 56 (1976), T330-331.

[5] M.S.Hecht and J.D.Ullman, Characterizations of Reducible Flow Graphs, J. of the ACM 21 (1974), 367-375.

[6] R.M.Karp, Reducibility among Combinatorial Problems, in R.E. Miller and J.W. Thatcher (eds.), Complexity of Computer Computations, Plenum Press, New York, 85-103, 1972.

[7] V.N.Kasyanov, Some Properties of Fully Reducible Graphs, Inf. Proc. Letters 2 (1973), 113-117.

[8] A.Shamir, A Linear Time Algorithm for Finding Minimum Cutsets in Reducible Graphs, SIAM J. Computing 8 (1979), 645-655.

[9] R.E.Tarjan, Testing Flow Graph Reducibility, J. Comp. Systems Sci. 9 (1974), 355-365.

RESOLUÇÃO DE EQUAÇÕES - Uma metodologia de estudo

Tiarajú Asmuz Diverio
Instituto de Informática PUC RS
Cx.P.1429 90620 P Alegre RS

Resumo:

Neste trabalho é proposto uma metodologia para o estudo de métodos de equações algébricas e transcendentes, através de fichas Instrucionais de Métodos, a qual conteria todas as informações necessárias sobre aplicabilidade, convergência, eficiência, descrição, casos patológicos e informações sobre implementações.

INTRODUÇÃO

Em muitos livros de cálculo numérico e análise numérica, encontram-se diversos métodos de resolução de equações. O tempo vem e vai e surgem novos livros de cálculo numérico, apresentando métodos "novos", sendo que alguns, na realidade estão sendo redescobertos. Vê-se, também, a falta de metodologia e de organização nesta área, pois não se tem catálogo, ou lista de métodos existentes, com uma quantidade significativa de informações sobre cada método.

Neste trabalho é proposta uma ficha Instrucional de caracterização dos métodos de resolução de equações algébricas e transcendentes, com a qual se pretende estabelecer uma abordagem, uma filosofia de estudo de métodos. É apresentado um exemplo de ficha instrucional para o método de Newton-Raphson.

A FICHA INSTRUCIONAL

A ficha instrucional é composta de vários itens que, no seu conjunto, constituem todas as informações necessárias para as pessoas que queiram utilizar os métodos de resolução de equações, quer com ênfase da análise numérica, ou com ênfase prática de utilização do método, ou queiram implementar alguns métodos, ou ainda, fazer considerações de carater computacional.

Os itens que constituem a ficha instrucional de um método são: nome, classificação, objetivo, uma descrição, a interpretação geométrica, limitações e restrições do método, o erro no método, a ordem de convergência, os índices de eficiência, o algorítmo, implementações existentes, a listagem do programa, mensagens de erro, dados de entrada e saída, exemplos, bibliografias e observações.

Nome do método

O nome do método como é mais conhecido na bibliografia.

Classificação

Quanto a natureza, os métodos podem ser:

a) Métodos de Quebra, que são métodos onde a idéia básica consiste de, a partir de um intervalo fechado I:=[a,b], que contenha uma raiz da função f(x) (contínua no intervalo), determinar um ponto xi que "quebre" o intervalo, de modo que a raiz pertença a um dos subintervalos formados;

b) Métodos Iterativos, eles envolvem o conceito de iteração (aproximação sucessiva), que é um dos conceitos mais importantes usados em métodos numéricos para resolução de diversos problemas matemáticos. Em um método iterativo partimos de um valor inicial x0, e construímos uma seqüência de aproximações gradualmente precisas, até que se satisfaça algum critério de parada. Se existe convergência, existe limite de seqüência, que será a raiz da equação, no caso de resolução de equações. Um passo do método iterativo pode calcular novo valor para um ou mais pontos, reusando ou não informações anteriores, com o que se pode classificar os métodos iterativos em unipontuais ou multipontuais, podendo ambos ser com ou sem memória.

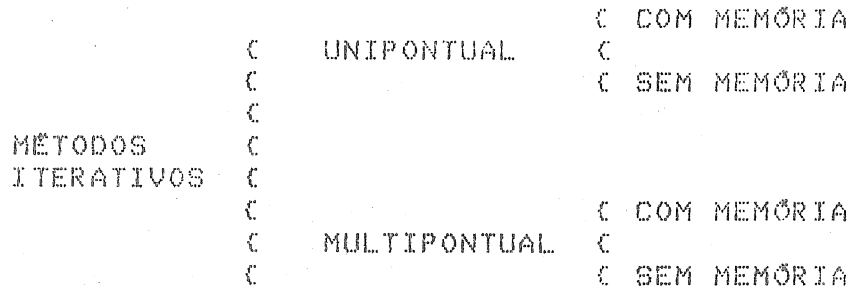


Fig. 1 Classificação dos métodos iterativos

c) Métodos intervalares são métodos que, em vez de determinar seqüência de números, determinam seqüência de intervalos que convergem para a solução. Para isso, tem-se que ter toda aritmética intervalar;

d) Métodos híbridos. Um método híbrido consiste em combinar mais de um método para a determinação de uma raiz. Eles podem ser combinados, ou aplicados isoladamente.

Objetivo

Uma descrição da utilidade do método; em que casos é aconselhável o uso, como o cálculo de raízes reais de um polinômio, ou cálculo de raízes complexas, ou para raízes múltiplas de um polinômio, ou ainda, para o cálculo de raízes reais de equações transcendentes. Isto serve para orientação de um usuário quanto a que método tem que usar para resolver o seu problema.

Descrição

Neste ítem serão dadas as condições em que o método deverá ou poderá, com maior exito, ser usado, contendo um resumo descritivo e as fórmulas utilizadas.

Interpretação geométrica

O gráfico que dá a interpretação do método, para que se tenha o entendimento de como funciona e quando converge o método.

Limitações e restrições

Neste ítem são dados casos de não convergência do método, bem como, situações que alteram o desempenho normal do método.

Erro no método

A determinação de erro, o cálculo do erro é algo da análise numérica, sendo muitas vezes um trabalho árduo. Neste item, não se tem por finalidade a sua determinação, ou como foi calculado, simplesmente se dará a fórmula do erro, que pode ser encontrado na bibliografia recomendada. Se dará o resultado da determinação do erro e onde pode ser encontrada.

Ordem de convergência

Seja (Xi) := X0, X1, ... uma seqüência de valores obtidos a partir de uma função de iteração ø(x), a qual converge para um valor a, tal que f(a)=0. Seja ainda, en o erro na iteração Xn, que é en = | Xn - a |.

Se existe uma constante C ≠ 0, e um número p tal que:

$$e_{n+1} := C\, e_n^{\,p} \qquad e \qquad \lim_{n \to \infty} \left(\frac{e_{n+1}}{e_n^{\,p}} \right) = C$$

então p é chamado de **ordem de convergência** da seqüência (ou método) e C é a **constante assintótica do erro**.

Para p = 1, 2, ou 3 a convergência é dita linear, quadrática ou cúbica, respestivamente.

A ordem de convergência é uma medida que indica qual é o ganho em precisão em uma iteração. Supondo C = 1, e que Xn coincide com a solução exata em k dígitos, então Xn+1 se aproxima de a em p.k dígitos significativos.

Índice de eficiência

Para se estudar complexidade computacional de métodos, algumas medidas de eficiência de métodos iterativos são apresentadas, com as quais poderá se comparar os diferentes métodos iterativos.

Para melhor caracterização da função de iteração ø, e com a finalidade de determinar o custo de uma iteração, introduz-se os parâmetros: **e** - o número de funções avaliadas por iteração (isto é, f, f', f'', ...); **m** - o número de valores reusados na iteração (isti é, xi-1, xi-2, ...). Seja ainda, **p** a ordem de convergência vista no item anterior.

Os índices E1 e E2 são conhecidos como o índice de eficiência de TRAUB e o de OSTROWSKI e são dados pelas fórmulas:

$$E1 := p/e \quad , \quad E2 := p^{(1/e)}$$

como é visto em [TRA64].

Estes índices são medidas assintóticas, elas caracterizam um particular método iterativo quando o número de iterações tende ao infinito. Quanto maior o valor de Ei, mais eficiente assintoticamente é o método iterativo. Estas medidas são aplicáveis somente na vizinhança da raiz **a**, o que resulta, que elas não são sempre adequadas para tratar com problemas práticos.

Uma medida eficiente ideal e para ser útil à comparação entre métodos deve ser definida em função do número de avaliações de funções e derivadas por iteração e o custo destas avaliações; do custo do cálculo da função, que é a combinação dos custos dos elementos da função de iteração; da ordem de convergência **p**; e do tamanho da constante assintótica do erro.

Em função disto, FELDSTEIN e FIRESTONE, em [FEL69], sugeriram a medida:

$$E3 := p^{(1/H)} \quad , \text{onde } H := e.(1+A/B).$$

Sendo p e **e** a ordem de convergência e o número de funções avaliadas, como já definido. E **A** é o número de operações aritméticas necessárias em uma iteração, sem incluir a avaliação da função ou derivadas. **B** é o número das operações requeridas para avaliar a função e derivadas.

Outra medida foi sugerida por PATERSON, em [PAT72], como:

$$E4 := (1/M).\log_2 p$$

onde M é o número de divisões e multiplicações requeridas para o cálculo da função de iteração.

Já KUNG e TRAUB, em [KUN73], sugeriram a medida E5, como:

$$E5 := \frac{\log_2 p}{\sum ni \cdot V(f^{(i)}) + c(\phi)}$$

onde, ni é o número de avaliações de $f^{(i)}$ usados na função de iteração ϕ; $V(f^{(i)})$ é o número de operações aritméticas requeridas para uma avaliação de $f^{(i)}$; $c(\phi)$ é o número de operações aritméticas requeridas para combinar $f^{(i)}$ para formar a função de iteração ϕ.

Considerando que maior exatidão na solução implica em maior número de iterações, mais a idéia da convergência e a relação base produtividade por custo, levaram SCHÄFFER ([SCH66]), a sugerir um novo índice de eficiência:

$$E6 := \frac{C \cdot p}{|\phi'(x0)| \cdot (A+B) \cdot ASC}$$

onde: C é a constante de normalização conveniente; p a ordem de convergência; ASC o número de algarismos significativos corretos que se deseja obter; $|\phi'(x0)|$ o valor absoluto da primeira derivada do método avaliada no ponto inicial x0; A é o número de operações aritméticas necessárias em uma iteração, sem incluir a avaliação da função ou derivadas e B é o número de operações necessárias para avaliar a função e derivadas.

Existem outras medidas de eficiência, mas o estudo destas medidas foge ao objetivo deste trabalho, podendo entretanto, ser acrescentadas nas fichas instrucionais.

Algorítmo

Deve ser dado um algorítmo em uma linguagem lógica, ou seja, de descrição, caracterizando as etapas da resolução do problema pelo método em questão. Este algorítmo deve conter testes e verificações de convergência, de exatidão e se satisfaz os critérios de parada.

Implementações

Este item diz respeito a uma tabela de referência a implementações existentes, deste método, em equipamentos usuais, contendo o nome do programa, ou rotina e a linguagem de programação que foi implementado.

Listagem do programa

A listagem do programa (fonte) de alguma das implementações que estejam disponíveis nos equipamentos usuais.

Mensagem de erro

Deve ser dada a relação das mensagens de erro, que podem ocorrer neste método, caracterizando a gravidade e, se possível, como contornar.

Dados de entrada e saída

Todas as possíveis informações necessárias para que se utilize o programa, qual resultado e de que forma são emitidos.

Exemplos de utilização do programa

Exemplos que caracterizam a utilização do método, pelo programa da listagem, não só para ilustração e teste do programa, mas também para base de aprendizagem de usuários.

Bibliografia

Consta de uma relação de referências bibliográficas de livros de cálculo numérico, análise numérica e métodos computacionais, que contenham informações do método em questão.

Observações

Nas observações, devem ficar informações que não se encaixem nos itens anteriores, mas que se ache importante referir.

UM EXEMPLO DE FICHA INSTRUCIONAL

Será apresentado a seguir um exemplo de ficha instrucional para o método de resolução de equações algébricas e transcendentes, extraído do Manual Instrucional do Usuário do SINAI-16 ([DIV86a]).

S I N A I - 1 6
Ficha Instrucional do Método
Resolução de Equações Algébricas

1. MÉTODO: **Newton-Raphson**

2. CLASSIFICAÇÃO:

... Método de Quebra	.X. Unipontual SEM memória
.X. Método Iterativo	... Unipontual com memória
... Método Intervalar	... Multipontual sem memória
... Método Híbrido	... Multipontual com memória

3. OBJETIVOS:

- .X. Cálculo de raízes reais de polinômios
- .X. Cálculo de raízes reais de equações transcendentes
- ... Cálculo de raízes complexas
- ... Cálculo de raízes múltiplas

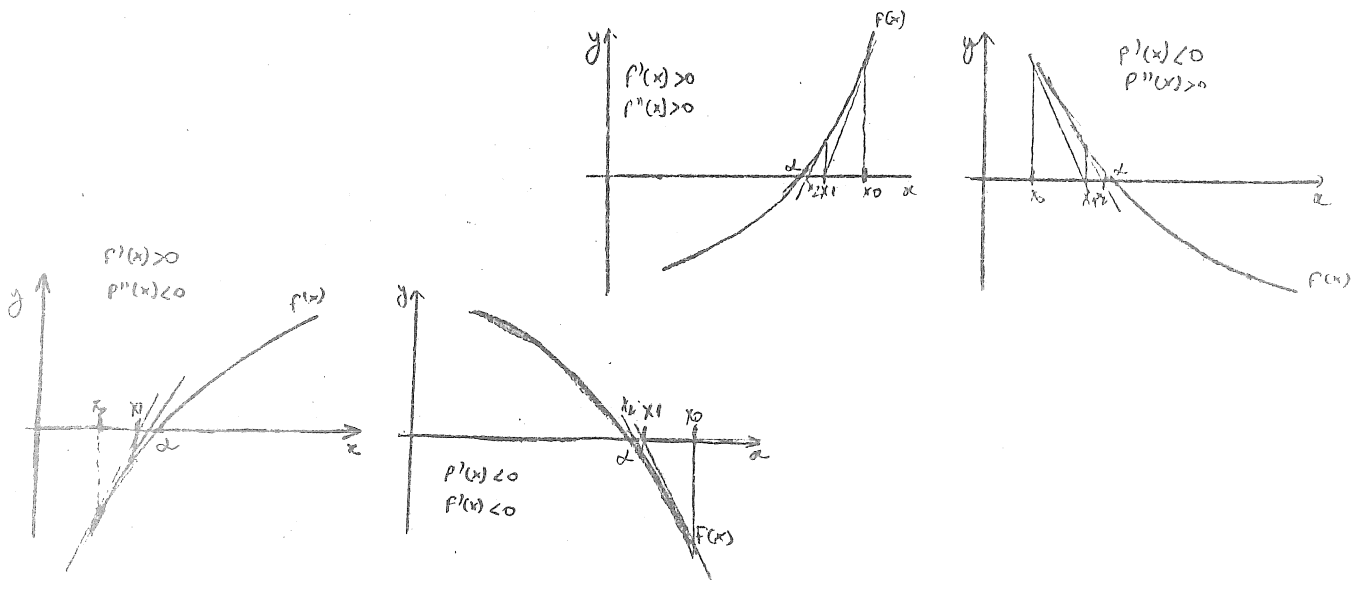
4. DESCRIÇÃO:

O método de Newton-Raphson é essencialmente um método iterativo, onde a função de iteração do método ø(x) é dada pela fórmula:

$$ø(xn) := xn+1 := xn - f(xn)/f'(xn), \text{ com } f'(xn)\neq 0$$

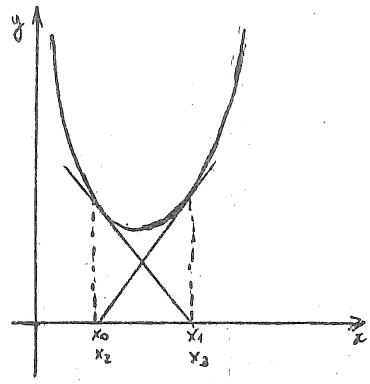
O método de Newton necessita uma estimativa inicial x0. Em cada iteração são avaliadas a função e a derivada. É bom lembrar que a derivada é a declividade da reta tangente no ponto, e portanto, a intersecção do eixo x, com a reta tangente no ponto xn, determina o ponto xn+1.

5. INTERPRETAÇÃO GEOMÉTRICA:

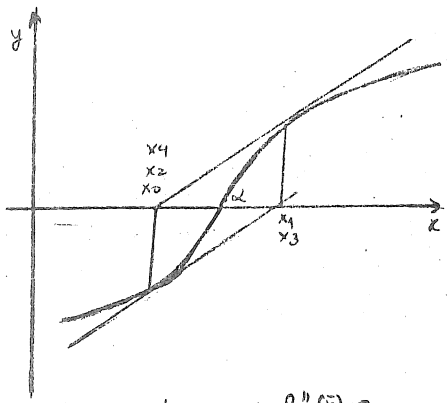


6. LIMITAÇÕES E RESTRIÇÕES:

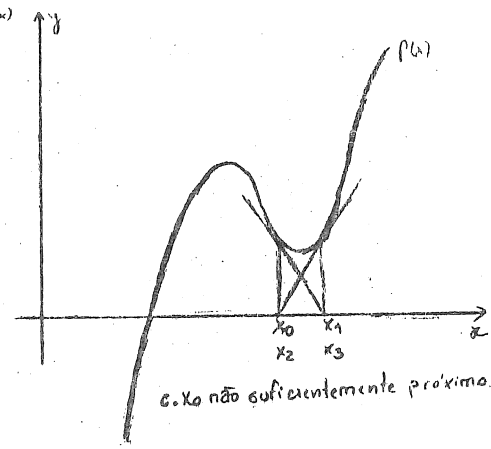
Algumas vezes o método de Newton não converge, isto não ocorre somente quando f'(x)=0; mas o método não converge e fica oscilando indefinidamente. Estes casos são quando não há raiz real, ou quando há simetria de f(x) em torno da raiz a, que implica em multiplicidade impar, maior ou igual a três: ou ainda, a estimativa inicial x0 estiver tão distante da raiz, que certa parte da função "prenda" a iteração, (devido a existência de mínimo local positivo ou máximo local negativo), como mostrado nas figuras abaixo:



a. nenhuma raiz real



b. Derivada segunda $P''(x)=0$. raiz múltipla.



c. Xo não suficientemente próximo.

Para que haja convergência, tem-se ainda que, f''(x) não se torne excessivamente grande e que f'(x) muito pequena, próximo de zero.

7. ERRO DO MÉTODO:

$$e_{n+1} = \frac{f''(c)}{2.f'(xn)} * e_n^2 \quad , \text{ onde c está entre xn e xn+1}$$

8. ORDEM DE CONVERGÊNCIA:

O método de Newton é de convergência quadrática, ou seja p=2 para raízes simples, perdendo este desempenho para raízes múltiplas. (Isto dentro de certa proximidade da raiz).

9. ÍNDICE DE EFICIÊNCIA:

Índice de		valor
TRAUB	E1	1.000
OSTROWSKI	E2	1.414
FELDSTEIN	E3	1.414
PATERSON	E4	1/2 . n
KUNG-TRAUB	E5	1/4 . n
SCHÄFFER	E6	

10. ALGORÍTMO:

```
INICIO
   K:=0                              (k - Número de iterações)
   LEIA( x0 , L, D )                 (x0 - valor inicial,
                                      L - limite de iterações,
                                      D - dígitos corretos desejados)
   ENQUANTO ( ASC<D ou K≠L ou FLK<min )          (min=10^-(D+1))
   FAÇA INICIO
          K:=K+1;
          FK:=f(Xk);
          FLK:=f'(Xk);
          SE FLK ≠ 0
             ENTÃO Xk+1:= Xk - FK/FLK
                   ASC;
   FIM;
   SE FLK < min              (f'(Xk)=0)
     ENTÃO ESCREVA ( Xk, FK, " derivada é zero")
           ESCREVA ("não dá para usar Newton com este valor");

     SENÃO SE ASC < D
              ENTÃO ESCREVA (Xk+1, ASC, K, "não convergiu a",D)
              SENÃO ESCREVA ( Xk+1, ASV, K);
FIM.
```

11. IMPLEMENTAÇÕES:

Equipamento	Nome do Programa	Linguagem
HP85	TIAPOLI	BASIC
LAB8037	LMCREAT	PASCAL
MAXXI	NEWTON	BASIC
I7000PCxt	SINAI-16	BASIC

12. LISTAGEM DO PROGRAMA:

Veja no Manual de Manutenção do Programa do SINAI-16, nas linhas de 18350 a 20100.

13. MENSAGENS DE ERRO:

Xk, FK, "derivada é zero"; "não dá para usar Newton com este valor"; A derivada primeira é nula, e portanto o método não pode ser aplicado, por ocasionar divisão por zero. Se isto ocorreu com valor inicial, tem-se que entrar com outro valor inicial, mas se ocorreu durante o processo, terá que ser utilizado outro método.

Xk+1, ASC, K "não convergiu a ", D; O método não conseguiu produzir a solução com a exatidão de D algarismos significativos corretos em K iterações. É dada a melhor aproximação obtida, com sua respectiva exatidão.

14. DADOS DE ENTRADA E SAÍDA:

ENTRADA: x0 - valor inicial
L - Limite de iteração
D - Dígitos corretos desejados

SAÍDA: Raiz procurada Xk+1
Algarismos Significativos Corretos no resultado ASC
Número de iterações calculadas

Saída Opcional:
Uma tabela contendo os principais valores parciais, para uma boa compreensão do método, contendo o número de iterações, a aproximação Xk, o valor da função na aproximação f(xk), o valor da derivada na aproximação f'(Xk) e o valor do ASC, algarismos significativos corretos.

15. EXEMPLOS DE EXECUÇÃO:

Calcular a raiz do polinômio $Pn(x) = x^4 + 2x^3 - 7.5x^2 - 20x - 11$ com o valor inicial x0=-0.5, com no mínimo 5 algarismos significativos corretos (ASC>5), e com x0= 3.0 e ASC>5.

No inicio do programa:

```
DEFINIÇÃO DA FUNÇÃO

3  ( alterar função)

Entre com o grau do Polinômio (1 a 20):? 4

Entre com os coeficientes an...a0

a4 : ? 1
a3 : ? 2
a2 : ? -7.5
a1 : ? -20
a0 : ? -11
```

```
        Definição da função

        Função atual                1.0000000 x4
                                    2.0000000 x3
                                   -7.5000000 x2
                                  -20.0000000 x1
                                  -11.0000000 x0


        1. Função Correta

        2. Alterar Coeficientes

        3. Alterar função

1 ( Função Correta)

              Resolução de Equações Polinômiais


        1. Definição da função
        2. Gráfico da função
        3. Cálculos das Cotas
        4. Cálculo das raízes reais
        5. Cálculo das raízes complexas
        6. Deflacionar raízes
        7. Voltar a tela-chave de primeiro nível

4 ( Cálculo das raízes reais)

              Cálculo das raízes reais


        A. Método da Bissecção
        B. Método de Newton
        C. Método da Secante
        D. Método de Newton para raízes múltiplas
        E. Método Híbrido C
        F. Método MIDREM
        G. Voltar a tela-chave de segundo nível

B (Método de Newton)

              Método de Newton

     Mostrar os resultados parciais (S/N) ? S

     Entre x0 e ASC ? -0.5,5

< resultado >
```

I	X	F(X)	F'(X)	ASC
0	-0.5000000	-3.0625000	-11.5000000	0.0
1	-0.7663043	-0.6332331	-6.7820646	0.0
2	-0.8596731	-0.0738070	-5.2119997	1.5
3	-0.8738341	-0.0016451	-4.9799620	3.1
4	-0.8741645	-0.0000001	-4.9745705	7.3

Fazer outro calculo (S/N) ? S

Mostrar os resultados parciais (S/N) ? N

Entre x0, ASC ? 3.0,5

<resultado>

aproximação inicial: 3

Numero de iteraçÕes: 3
Raiz: 3.03524499
ASC: 6.743843

Fazer outro calculo (S/N) ? N

<volta a tela-chave de terceiro nível>

0 <voltar a tela-chave de segundo nível>

7 <voltar a tela-chave de primeiro nível>

C < fim >

16. BIBLIOPGRAFIA:

[STA79] [BAR83] [RAL65] [FOR77] [DOR79] [SH084] [TRA64]
[MAR82] [CLA83] [KRO79] [VAN78] [DEW83] [JOH82] [CON72]

17. OBSERVAÇÕES:

COMENTÁRIOS FINAIS

As fichas Instrucionais dos métodos contem um conjunto completo de informações sobre o mesmo, e o conjunto delas propiciam a sua catalogação, constituindo uma forma racional, útil e inteligente para catalogação dos métodos existentes.

As fichas instrucionais podem, também, ser úteis ao desenvolvimento de métodos híbridos, que pela combinação de um ou mais métodos, pode-se conseguir melhores resultados em termos de ordem de convergência e de índice de eficiência.

As fichas Instrucionais são um instrumento dinâmico, necessitando testes e atualizações periódicas, a medida que surgem novas informações , como por exemplo: novo índice de eficiência, que a elas podem ser acrescentadas.

Estas Fichas constituem uma proposta para a definição de uma metodologia no estudos de métodos computacionais de resolução de equações.

REFERÊNCIA BIBLIOGRÁFICA

[BAR83] BARROSO,L.C. et alli. Cálculo numérico. São Paulo, Harper & Row, 1983.

[CLA83] CLAUDIO,D.M. & ROYO DOS SANTOS,J.A. Microcomputadores e minicalculadoras programáveis. seu uso em ciências e engenharia . São Paulo, Edgar Blücher, 1983.

[CON72] CONTE,S.D. & DE BOOR,C. Elementary numerical analysis an algorithmic approach. New York, McGraw Hill, 1972.

[DEW83] DEW,P.M. & JAMES,K.R. Introduction to numerical computation in PASCAL. Hong Kong, Macmillan Press, 1983.

[DIV86] DIVERIO,T.A. Software numérico aplicativo e instrucional. Porto Alegre, CPGCC da UFRGS, 1986. (dissertação de mestrado)

[DIV86a] DIVERIO,T.A. SINAI-16 = Manual Instrucional do Usuário. Porto alegre, CPGCC da UFRGS, 1986.

[DIV86b] DIVERIO,T.A. SINAI-16 = Manual de Manutenção do Programa. Porto Alegre, CPGCC da UFRGS, 1986.

[DOR79] DORN, W.S. & McCRACKEN,D.D. Cálculo numérico com estudos de casos em FORTRAN IV. Rio de janeiro, Campus, 1979.

[FEL69] FELDSTEIN,A. & FIRESTONE, R.M. A study of Ostrowski efficiency for composite iteration algorithms. Proceedings, 1969.

[FOR77] FORSYTHE,G.E et alli. Computer methods for mathematical computations. Englewood Cliffs, Prentice Hall, 1977.

[JOH82] JOHNSTON,R.L Numerical methods a software approach. New York, John Willey & Sons, 1982.

[KRO79] KRONSJO, L. Algorithms their complexity and efficiency. Chichester, John Wiley & Sons, 1979.

[KUN73] KUNG,H.T & TRAUB,J.F Optimal order of one-point and multi-point iteration. Journal of ACM, New York, 21(4): 643-51, oct. 1974.

[MAR82] MARINS, J.M Métodos computacionais para o cálculo de raízes de equações polinomiais. Porto Alegre, CPGCC da UFRGS, 1982. (dissertação de mestrado).

[PAT72] PATERSON, M.S. Efficient iterations for algebric numbers in complexity of computer computations. London, Plenum Press, 1972.

[RAL65] RALSTON,A. A first course in numerical analysis. New York, McGraw Hill, 1965.

[SCH86] SCHAFFER, M.I. Análise computacional de métodos iterativos pontuais e multipontuais na resolução de equações. Porto Alegre, CPGCC da UFRGS, 1986. (dissetação de mestrado).

[SHO84] SHOUP,T.E Applied numerical methods for the microcomputer. Englewood Cliffs, Prentice Hall, 1984.

[STA79] STARK, P.A. Introdução aos métodos numéricos. Rio de Janeiro, Interciência, 1979.

[TRA64] TRAUB, J.F. Iterative methods for the solution of equations. Englewood Cliffs, Prentice Hall, 1964.

[VAN78] VANDERGRAF, J Introduction to numerical computation. New York, Academic Press, 1978.

UNA VARIANTE DEL SISTEMA DE ROTORES

ERNESTO AZORIN y NELSON BALOIAN

Departamento de Ciencias de la Computación

Universidad de Chile

Casilla 2777

Santiago, Chile.

Resumen En este trabajo se describe una variante de un sistema criptográfico basado en el principio de los rotores. A la concepción original de este sistema se le han introducido modificaciones que hacen dependiente de la clave a la función de transformación; contempla, además, el uso de variables pseudoaleatorias en el algoritmo de rotación de los rotores. Esto vuelve inútiles los métodos tradicionales de ataque para rotores clásicos.

1- Introducción

Los rotores son un sistema criptográfico de sustitución polialfabética. Esto significa que cada letra x del texto originial se sustituye por otra c aplicándole una transformación $f(x) = c$. Además, a cada letra del texto original se le aplica una función distinta; es así como una misma letra se traduce, en general, de manera diferente en cada aparición.

En su concepción original, una máquina de rotores consiste en una batería de t rotores o ruedas. El perímetro de cada rotor R_i tiene$n$ contactos eléctricos en cada una de sus caras, uno por cada letra del alfabeto ($n = 26$ en el caso del alfabeto inglés). Cada contacto de la cara anterior está conectado por un cable a otro de la cara posterior haciendo corresponder uno a uno los contactos de ambas caras. Queda así definida una función de transformación f_i para cada rotor R_i . Cada rotor puede girar para colocarse en una cualquiera de n posiciones distintas y alterar la función f_i . Formalmente se puede escribir que cuando el rotor R_i está en la posición l_i la función de transformación resultante se redefine :

$$F_i = (f_i (x - l_i) \bmod n + l_i) \bmod n \qquad (1.1)$$

Los t rotores van colocados de modo que los contactos de la cara posterior del i-ésimo rotor estén en contacto con los de la cara anterior del rotor en la posición i+1. Para transformar una letra (por ejemplo la j-ésima del alfabeto) se envía una señal eléctrica por el contacto que le corresponde a su posición en la cara anterior del primer rotor. La señal viaja a través de los rotores en sucesión y emerge por alguno de los contactos de la cara posterior del último, lo cual da la letra del texto cifrado correspondiente. A la letra del texto original se le aplican, entonces, t transformaciones sucesivas :

$$c = F(x) = F_1(F_2 (\ldots F_{t-1}(F_t (x)))) \qquad (1.2)$$

Después de haber cifrado una letra, uno o más rotores cambian de posición originándose una nueva función de transformación F.

Un sistema de rotores famoso es el **Enigma** [Denn83] usado por los alemanes en la segunda Guerra Mundial. En el **Enigma**, después de toda transformación de una letra sigue un avance de una posición del primer rotor . Una vez que completa una rotación entera, el segundo avanza una posición. Cuando el segundo completa una vuelta, el siguiente avanza un lugar y así sucesivamente con los otros. La clave que mantiene el secreto del texto la constituye la posición inicial de los

rotores y las conexiones en cada uno de ellos. Formalmente, la clave de este sistema está constituida por las funciones de transformación de cada rotor y la posición inicial de ellos.

Para descifrar un texto hay que enviar los caracteres cifrados a través de los rotores pero ahora en sentido inverso, es decir, la señal que corresponde al carácter cifrado entra por un contacto de la cara posterior y sale por el de la cara anterior del rotor al cual está asociado. Esto corresponde a aplicarle la función inversa f_i^{-1} del rotor R_i. Si llamamos c_i al carácter cifrado por la función F_i,

$$c_i = F_i(x) \tag{1.3}$$

y F_i^{-1} se escribe como:

$$F_i^{-1}(c_i) = (f^{i-1}(c_i - l_i) \bmod n + l_i) \bmod n = x \tag{1.4}$$

Para obtener el carácter original es necesario que los rotores estén en la misma posición l_i en la que aquél fue cifrado. Si los caracteres se descifran en el mismo orden en que fueron cifrados basta generar las posiciones de la misma manera que en el proceso de cifrado.

En este trabajo se enfrenta el diseño de un sistema de rotores para proveer una herramienta que permita cifrar textos y programas con las siguientes características :

- Las funciones de transformación f_i de los rotores y la posición inicial de cada uno ellos se derivarán de la clave con que se cifre el texto.

- Las posiciones siguientes de los rotores se darán de acuerdo a un algoritmo de generación de variables pseudoaleatorias, que dependerá también de la clave.

Es bien conocida la falta de aleatoriedad propia de los generadores congruenciales lineales **[Frei84]**. Por otro lado, el interés por construir sistemas criptográficos ideales ha suscitado la búsqueda de secuencias pseudoaleatorias criptográficamente seguras ([Blum84], **[Blum86]**, **[Gold86]**, **[Sham83]**). El uso de la clave para determinar el movimiento de los rotores provee una forma sencilla de modificar sucesivamente los ciclos propios de generadores congruenciales lineales, mejorando así las repeticiones "aleatorias" de dígitos. En este estudio se recurre a transformaciones de la clave original, análogas a las utilizadas por el DES [NBS77] , para calcular las claves parciales asociadas a los rotores.

2- La función de transformación

Llamemos A al alfabeto del texto que se quiere cifrar y sea n su cardinalidad. Sea x un número entre 1 y n que representa a la **x-ésima** letra del alfabeto. La función de transformación f_i de cada rotor debe ser tal que transforme x en un número $c = f_i(x)$ tal que c quede entre 1 y n para así poder asociarlo a otra letra del alfabeto. Además se debe cumplir que para un x' distinto a x, $f(x')$ debe ser distinto a $f_i(x)$ para todo x' ya que así se simulan las conexiones que asocian a cada contacto de la cara anterior de un rotor con otro único de la cara posterior. Para lograr esto se propone definir la función f_i como :

$$f_i(x) = (a_i * x) \bmod n + 1 \qquad (2.1)$$

en que : a_i es un número entero entre 1 y n que se deriva de la clave.

n es un número primo igual al número de símbolos que tiene el texto.

Es necesario que n sea primo ya que así cualquier a_i definido de la manera anterior cumplirá la condición de que el máximo común divisor entre a_i y n es 1. De aquí se desprende que para cada $x \in \{1,2,...,n\}$, $(a_i * x) \bmod n$ es un número distinto entre 0 y $n - 1$, lo que permite la asociación biunívoca x <--> $(a_i * x) \bmod n$.

Para abarcar la mayor cantidad posible de caracteres del código ASCII podemos utilizar un subconjunto de ellos de cardinalidad igual al primer número primo menor que 256, es decir 251.

Como en general, para dos rotores distintos R_i y R_j, se tiene que $a_i \neq a_j$, cada rotor aplicará una transformación distinta.

En este sistema se define la función de transformación **(2.1)** de cada rotor de modo que en el recorrido de ella nunca se encuentre el número cero (como consecuencia, tampoco hay un símbolo del alfabeto que sea representado por este número), ya que cuando apareciese, este valor podría propagarse hasta el final incluyendo información del texto original en el cifrado. Con la función de transformación así definida los valores del recorrido están en {1,...,n}.

3- El algoritmo de rotaciones

En un sistema de rotores $\{R_1, R_2, ..., R_t\}$ la cantidad de posiciones distintas que puede tomar el conjunto de ellos es n^t donde:

n es el número de posiciones posibles de cada rotor, es decir, el número de símbolos del alfabeto con que se trabaja (en este caso corresponde a 251).

t es el número de rotores en el sistema.

Como las funciones de transformación de cada rotor se mantienen fijas durante la codificación del texto, el número de sustituciones distintas que podemos generar con el sistema es precisamente el número de posiciones distintas del conjunto de rotores, es decir n^t .

Para dar mayor seguridad al cifrado del texto es deseable tener un número de sustituciones tan largo como el texto a codificar a fin de no repetir la misma sucesión de sustituciones. Podemos lograrlo haciendo lo siguiente:

- Se define un **estado de posiciones** como el conjunto de las posiciones de los rotores en un instante dado.
- Se enumeran todas los estados de posiciones posibles desde la posición 0 a la $n^t - 1$: cada estado de posiciones se representa por un número P de t dígitos en base n, es decir

$$P = p_1 p_2 \ \ldots \ p_{t-1} p_t \tag{3.1}$$

en el que cada p_i es un número entre 0 y $n - 1$ que representa la posición de cada rotor R_i en ese momento.

El valor en base diez del estado inicial de posiciones, P_0 , se calcula como:

$$P_0 = H * N_0 \ mod \ (n^t) \tag{3.2}$$

donde H, N_0 son números entre 0 y $n^t - 1$ derivados de la clave.

Los siguientes estados de posiciones , P_{i+1} , se calculan como :

$$P_{i+1} = H * N_{i+1} \ mod \ (n^t) \quad ; \quad N_{i+1} = N_i + 1 \tag{3.3}$$

Para encontrar las posiciones de cada rotor correspondiente al estado de posiciones P_i basta convertir este número a uno de base n y cada dígito dará la posición que le corresponde a cada rotor.

Como la divulgación o descubrimiento del período facilita el rompimiento del código se sugiere una modificación sencilla que esconde dicho período por medio de desfases sucesivos generados,

también, a partir de la clave. Para esto, cada D transformaciones se efectúa un salto de E posiciones en el valor de N_i :

$$N_i = (N_{i-1} + E) \bmod (n^t) \qquad (3.4)$$

donde D y E son números derivados de la clave de modo que D + E no sea divisor de n^t.

4- La clave

La clave K se ingresa como una palabra de $2k$ caracteres. Cada carácter se traduce a un número de tres dígitos correspondiente al valor en base diez de su representación binaria en ASCII, que da origen a una secuencia de $6k$ dígitos. Partiendo esta secuencia se construyen dos números, ci y cd, de $3k$ dígitos cada uno.

Obtención de los a_i :

Para derivar un a_i de los dos números ci y cd generados, seguimos la filosofía del sistema DES en el cual se efectúan varias permutaciones y desplazamientos circulares (indicados por RT en la figura 1) de los dígitos de ámbos números según una tabla predefinida. Luego, se escogen trés digitos de cada número ci, cd y, entremezclándolos, se construye un nuevo número m . Con éste se calcula

$$a_i = m \bmod (n-1) + 1 \qquad \text{(indicados por S en la figura 1)}$$

para obtener así un número entre 1 y $n-1$, que es la condición que deben cumplir los a_i.. Este proceso se vuelve a repetir las veces que sea necesario para obtener todos los a_i, $i = 1...t$ (uno para cada rotor).

Cada proceso de obtención de un a_i modifica los números generados a partir de la clave con lo cual a_{i+1} se obtendrá a partir de ci y cd distintos. Se asegura así que un mismo carácter se traduce, en general, de manera distinta en cada rotor como indica (2.1). Este procedimiento se esquematiza en la figura 1.

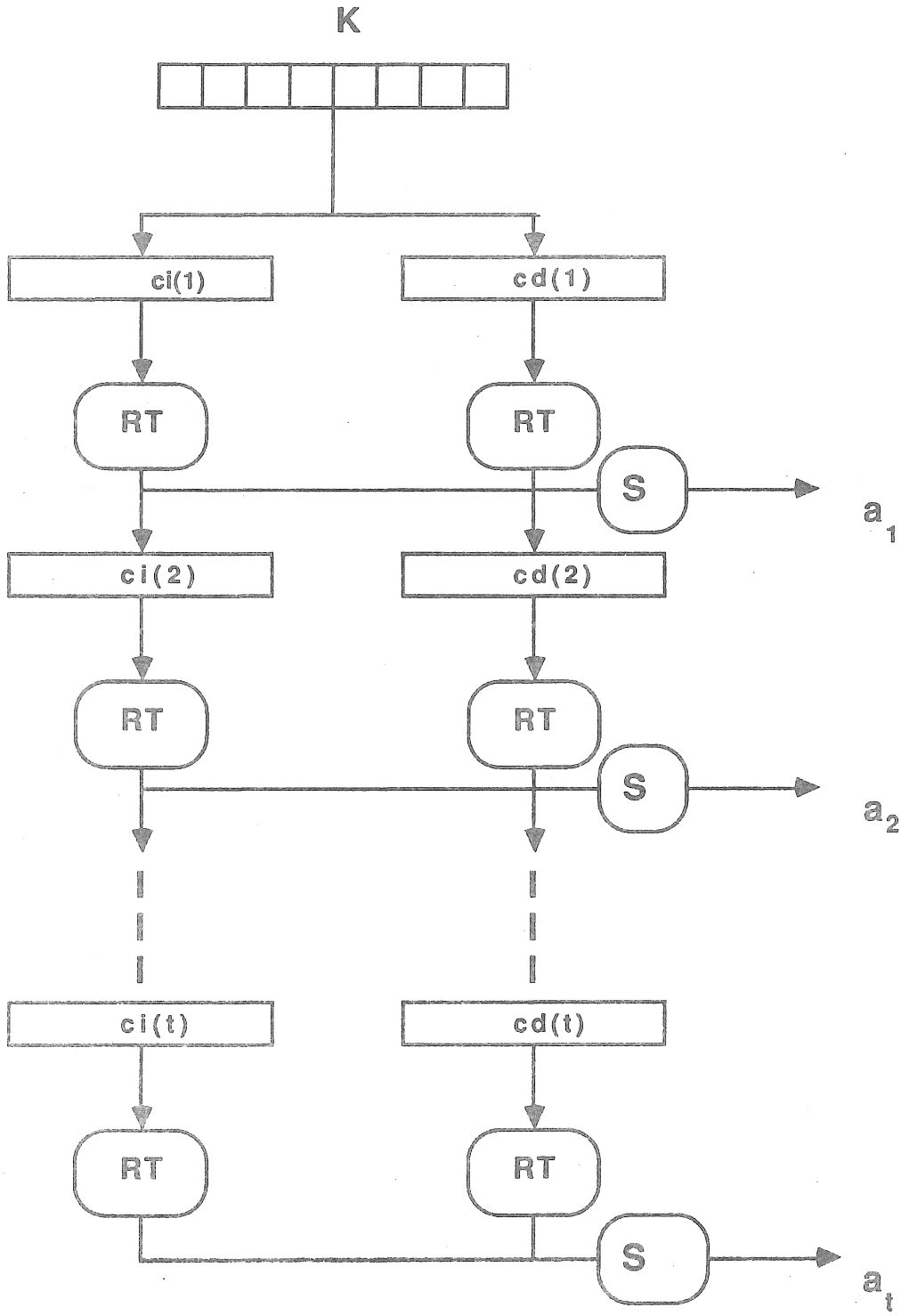


figura 1: Obtención de los a_i

Obtención de H :

H debe cumplir la condición

$$Mcd\ (\ H\ ,\ n^t\ )\ =\ 1 \qquad (4.1)$$

Por esto, el proceso que la genera obtiene un número m de manera semejante a la usada para generar los a_i a partir de los dígitos de ci y cd,
y calcula iterativamente valores Hi de la siguiente forma :
Se calcula primero H_1 :

$$H_1 = m\ mod\ (\ n^t\ ) \qquad (4.2)$$

Luego se comprueba si este número H_1 cumple la condición **(4.1)**. De no ser así, se busca el primero entre los siguientes mayores que él con el algoritmo :

$i = 1$
while $Mcd\ (\ H_i,\ n^t\ ) \neq 1$ *do*
begin
$H_i + 1 = (\ H_i + 1)\ mod\ n^t$;
$i = i + 1$;
end
$H = H_i$;

El número N_O que da los desplazamientos (posiciones) iniciales de los rotores se obtiene de la misma manera que H con la salvedad que no necesita cumplir la condición :

$$Mcd(\ N_O,\ n^t\ )\ =\ 1 \qquad (4.3)$$

Finalmente, los números D y E se pueden obtener de manera similar verificando que se cumpla:

$$Mcd(\ D + E,\ n^t\ )\ =\ 1 \qquad (4.4)$$

5 - Cifrado y descifrado

El procedimiento que cifra un archivo usa 3 parámetros : nombre de un archivo de entrada a cifrar, nombre de un archivo de salida donde quedará el archivo transformado, y una palabra clave.

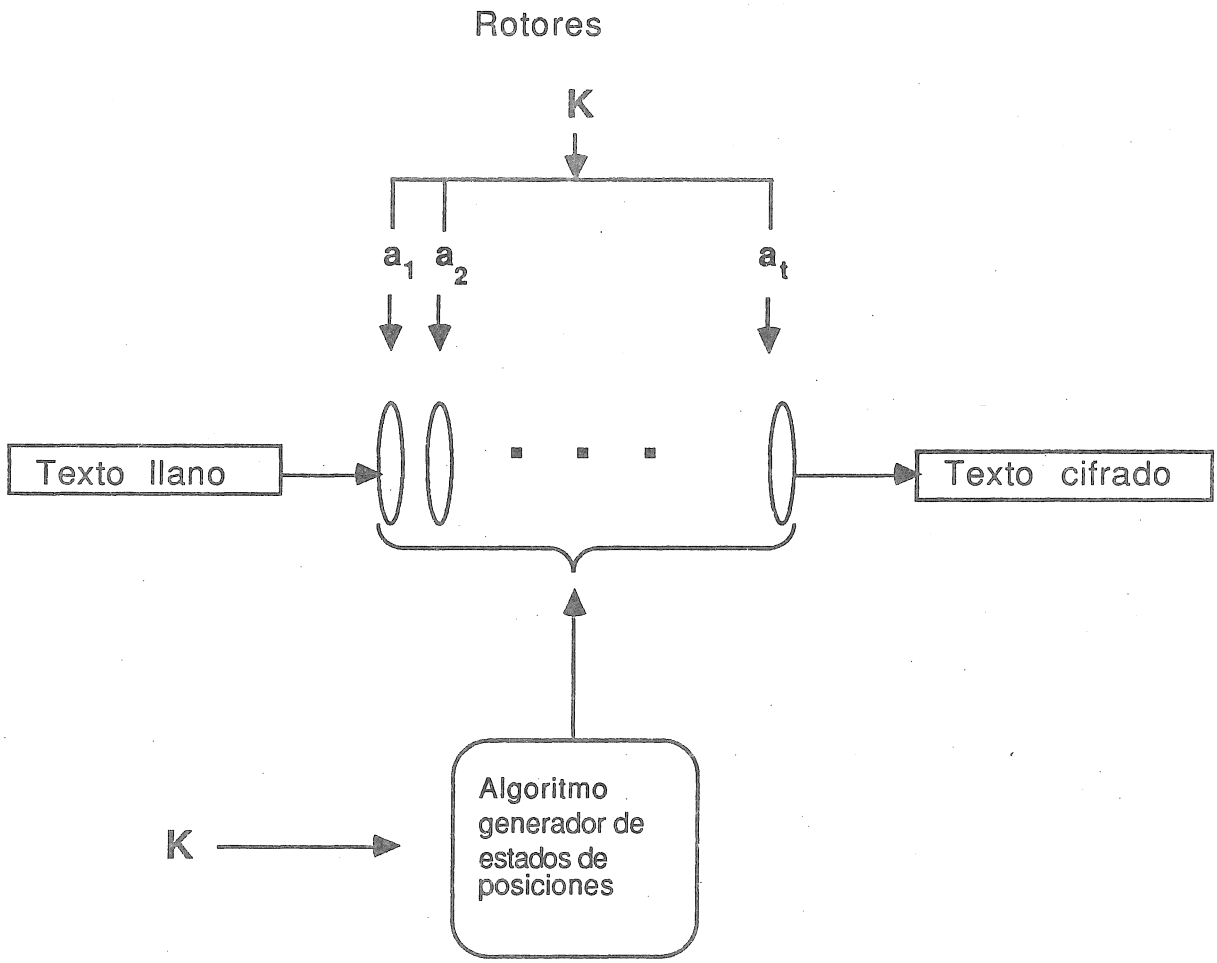


fig 2 Esquema del cifrado

Un subprograma se encarga de recibir la clave y generar los números que a partir de ella se derivan: a_i, $i = 1 \ldots t$; H ; N_O ; D y E . Luego, otro procedimiento aplica a cada carácter las t transformaciones correspondientes cada una al paso por un rotor. Una vez finalizadas estas transformaciones se escribe el cáracter resultante en el archivo de salida y se invoca un tercer subprograma que genera un nuevo estado de posiciones para el sitema de rotores por medio del algoritmo descrito anteriormente.

Para descifrar el texto, los parámertos necesarios son el nombre del archivo cifrado, el nombre del archivo de salida en el que se dejará el texto descifrado y la clave. El mismo procedimiento

empleado en el proceso de cifrado genera nuevamente los valores a_i , H , N_O , E y D. Con N_O se obtiene nuevamente la posición inicial de los rotores y con H se genera la misma secuencia de estados de posiciones usando el mismo algoritmo empleado en el proceso de cifrado.

En el proceso de descifrado la transformación que debe aplicársele a un carácter en cada rotor R_i es la inversa de la aplicada en el proceso de cifrado, es decir f_i^{-1} . Para obtener esta función inversa basta despejar x de la expresión **(2.1)**. Si llamamos c_i al carácter cifrado por $f_i(x)$ tenemos que :

$$f_i^{-1}(c_i) = ((c_i - 1) * inv(a_i, n)) \bmod n \qquad (5.1)$$

en que $inv(a_i, n)$ es un número d_i tal que $(a_i * d_i) \bmod n = 1$. Por esto, en el proceso de descifrado es necesario además calcular los valores d_i por medio de $d_i = inv(a_i, n)$. Esto puede hacerse recurriendo al algoritmo descrito en **[Denn83]**. El esquema del proceso de descifrado es análogo al de la figura 1 pero ahora el texto de entrada se introduce por el extremo derecho del sistema de rotores.

6 - Comentarios finales

A pesar que los rotores clásicos generan secuencias de transformaciones de período largo, se sabe que éstas son menos aleatorias de lo que podrían parecer. La variante presentada apunta a mejorar esta propiedad de aleatoriedad.

La clave, por su parte, se compone de un máximo de 8 caracteres y no se aceptan menos de 4, siguiendo de este modo, las características de las claves de entrada en UNIX **[Fili86]**. Esto da un total de claves posibles igual a 256^8 si se contemplan todos los símbolos ASCII para formar la clave. Podemos suponer, con mucha seguridad, que no se usarán más de 100 símbolos (26 mayusculas, 26 minúsculas, 10 dígitos, signos de puntuación y matemáticos). Por esto, una estimacíon más realista del número de claves posibles es $(100)^8 = (10)^{16}$. Esta cantidad dificulta suficientemente un ataque contra la clave. (Suponiendo que la elección de los caracteres es equiprobable -no hay preferencias por secuencias de caracteres con significado propio, por ejemplo-, la entropía para la clave es de $H_k = log_2 (10)^{16} \approx 53{,}152$).

El sistema de rotores descrito se encuentra funcionando en un computador NCR-Tower 32/600 bajo el sistema operativo UNIX, con un número de 4 rotores (t =4) con lo cual, el número de estados de posiciones distintas es 251^4 ; por lo tanto se tiene igual número de funciones de transformación F distintas.

Existe además una versión de este sistema, con 6 rotores, para PCs.

Referencias

Blum84 Blum, M. and Micali, S. "How to Generate Cryptographically Strong Sequences of Pseudo-random bits", SIAM J. of Comp., 13, pp. 850-864 (Nov. 1984).

Blum86 Blum, L., Blum M. and Schub, M. "A Simple Unpredictable Pseudo-random number Generator" SIAM J. of Comp., 15, pp. 364-383 (May 1986).

Denn83 Denning, D. E. R. "Cryptography and Data Security", Addison-Wesley (1983).

Fili86 Filipsky, A. and Hanko, J. "Making Unix Secure", Byte pp. 113-128 (April 1986).

Frei84 Freize, A.M.; Kannan, R. and Lagarias, J. C. "Linear Congruential Generators do not Produce Random Sequences", Proc. of the 25th. Symp. on Foundations of Comp. Sci., IEEE, N.Y., pp. 480-484 (1984).

Gold85 Goldreich, O.; Goldwasser, S. and Micali, S. "How to Construct Random Functions", J. of the Assoc. for Comp. Mach, 33, 4, pp. 792-807 (Oct. 1986).

NBS77 "Data Encryption Standard", Federal Information Processing Standard Publication 46, National Bureau of Standards, US Departament of Commerce, Washington, D.C (Jan. 1977).

Sham83 Shamir, A. "On the Generation of Criptographically Strong Pseudorandom Sequences", ACM Trans. Comp. Syst. 1,1 pp. 38-44 (Feb. 1983).

PROPOSICION DE UN LENGUAJE DE NIVEL INTERMEDIO PARA UN SISTEMA DE PRODUCCION DE COMPILADORES

A. Campos, Y. Eterovic, G. García, R. Santis
Depto. Ciencia de la Computación
P. Universidad Católica de Chile
Casilla 6177 - Correo 22
Santiago - CHILE

RESUMEN

Se presenta el diseño de una máquina objeto virtual y de su lenguaje de ensamble, basado en las características principales de los lenguajes de programación modernos de mayor uso, y orientado a facilitar la generación automática del generador de código intermedio en un sistema de producción de compiladores. En particular, se describe la organización de la memoria, la representación de los tipos de datos básicos y estructurados, el manejo de subprogramas y de múltiples procesos, y las características relevantes del lenguaje de ensamble.

1.- Introducción

El estado del conocimiento en el área de construcción de compiladores ha permitido desarrollar generadores automáticos para las diferentes etapas del proceso de compilación: análisis léxico, sintáctico y semántico, y generación de código. Estos generadores automáticos son programas computacionales que, a partir de una especificación formal de un lenguaje de programación, generan la etapa correspondiente de un compilador para ese lenguaje.

En particular, es posible construir con cierta facilidad generadores de analizadores léxicos y sintácticos, a partir de especificaciones basadas en expresiones regulares y gramáticas libres de contexto. Mayor dificultad presenta la construcción de generadores de analizadores semánticos y generadores de código, debido a la falta de formalismos apropiados para especificar la semántica de los lenguajes de programación y las características de las máquinas objetos para las cuales se desea generar código (Campos y Eterovic, 1988).

Un sistema de producción de compiladores es un conjunto de generadores automáticos para todas o algunas etapas del proceso de compilación, organizadas de modo de reducir la cantidad y complejidad de la programación necesaria para construir compiladores para diferentes lenguajes y máquinas objetos.

En este artículo se presenta el diseño de una máquina objeto virtual con un único procesador y de su lenguaje de ensamble. El propósito de este diseño es proporcionar a un sistema de producción de compiladores un lenguaje común de nivel intermedio (el lenguaje de ensamble) que facilite la generación automática del generador de código intermedio de los diferentes compiladores producidos por el sistema. El diseño está basado en las principales características de los lenguajes de programación modernos de mayor uso, como se describe en las siguientes secciones.

2.- Los Registros y la Organización de la Memoria

Una característica común a todos los lenguajes de programación modernos es la estructura de bloques, normalmente conseguida a través de subprogramas que pueden anidarse estáticamente y activarse recursivamente hasta niveles arbitrarios de profundidad. El modelo de implementación más apropiado para este tipo de estructuras está basado en el funcionamiento de stacks que crecen o decrecen cuando se inicia o termina la ejecución de un subprograma, y cuyos elementos son representaciones de los subprogramas que están siendo ejecutados (Tanenbaum, 1978).

Por otra parte, los lenguajes más recientes también proveen facilidades para definir en un mismo programa y ejecutar simultáneamente varios procesos independientes, cada uno de los cuales requiere de su propio stack de ejecución (USDD, 1983; Wirth, 1985). Finalmente, también es posible para el usuario, en la mayoría de los lenguajes modernos, manejar parte de la memoria disponible en forma dinámica, esto es, solicitar y liberar memoria durante la ejecución de los programas.

En base a estas consideraciones, se ha diseñado la organización de la memoria de la máquina virtual en la forma de un arreglo lineal de bytes de 8 bits de longitud, conceptualmente dividido en 3 áreas durante la ejecución de los programas:

- Un área central, o área de código, destinada al almacenamiento de los programas una vez traducidos al lenguaje de ensamble de la máquina. La máquina provee acceso a esta área a través del registro Program Counter (PC), que contiene la dirección de la instrucción que debe ser ejecutada a continuación de aquélla que está siendo ejecutada;

- Un área organizada en la forma de un conjunto de stacks, utilizada para asignar un stack a cada uno de los procesos simultáneamente activos. En estos stacks se almacenan los registros de activación de los procesos activos y de los subprogramas que están siendo ejecutados. La máquina provee acceso a esta área a través de los registros Stack Pointer (SP), que contiene la dirección del tope del stack del proceso que está siendo ejecutado (el stack vigente), y Activation Record (AR), que contiene la dirección base del registro de activación que está en el tope del stack vigente;

- Un área organizada en la forma de un heap, definida como un área de memoria común para todos los procesos activos, que permite implementar las facilidades de manejo dinámico de memoria presentes en varios lenguajes de programación (Jensen y Wirth, 1975). La máquina provee dos instrucciones para manejar esta área: Allocate almacena en el stack vigente la dirección a partir de la cual hay un determinado número de bytes disponibles; Deallocate libera un determinado número de bytes, a partir de una dirección almacenada en el stack vigente. En ambos casos, el número de bytes también se encuentra en el stack vigente.

Cada uno de los tres registros mencionados, además de un cuarto registro de libre disposición, Mark (M), tiene 32 bits de longitud. Como las direcciones de memoria se representan en 4 bytes consecutivos o en éstos registros, es posible direccionar 2^{32} bytes.

3.- Tipos de Datos Básicos y Estructurados

Los siete tipos de datos básicos provistos por la máquina objeto y su longitud en bytes son los siguientes: Byte (1 byte), Word (4 bytes), DoubleWord (8 bytes), QuadWord (16 bytes), Floating (5 bytes), DoubleFloating (9 bytes), QuadFloating (17 bytes). En los tres últimos casos, el primer byte es utilizado para representar un exponente y el resto para representar una mantisa. Este conjunto posee la flexibilidad suficiente para permitir la representación de la totalidad de los tipos de datos elementales presentes en los lenguajes de programación modernos: integer, real, character, boolean, etc (Pratt, 1984).

Por otra parte, la forma más simple de tipos de datos estructurados presente en los lenguajes de programación modernos es la de los arreglos homogéneos, en las cuales todos los elementos de un arreglo pertenecen a un mismo tipo de datos. La estructura de estos arreglos puede ser estática, si es definida en el momento de la compilación de un programa y no puede ser modificada durante su ejecución, semidinámica, si es definida en el momento de la activación de un subprograma y no puede ser modificada durante su ejecución, o dinámica, si puede ser definida y modificada en cualquier momento durante la ejecución de un programa (Ghezzi y Jazayeri, 1982). La máquina virtual propuesta permite la representación y proveer las instrucciones necesarias para el manejo de arreglos de estructura semidinámica.

En el área de stacks de la máquina, un arreglo es representado por la dirección de su descriptor (Eterovic y Fuller, 1985). Un descriptor de arreglo (AD) es una estructura de bytes que contiene la información que define la estructura del arreglo y que permite localizar un elemento determinado de éste. Esta información consiste en el número de dimensiones del arreglo, el número de elementos en cada dimensión, y el número de bytes ocupados por cada elemento. La información contenida en el arreglo propiamente tal es almacenada en el área de stacks, a continuación del descriptor. Las instrucciones provistas para el manejo de los arreglos son las siguientes:

- PushAD almacena en el stack vigente el descriptor de un arreglo;

- CopyAD copia en el stack vigente el descriptor de un arreglo cuya dirección está en el tope del stack;

- PushElement almacena en el stack vigente la dirección de un elemento de un arreglo, a partir de los valores de los índices que identifican al elemento y que están almacenados en el tope del stack vigente;

- ArrayCopy copia el contenido del arreglo cuya dirección está en el tope del stack vigente en el arreglo cuya dirección está a continuación del tope del stack. Esta instrucción no verifica que los descriptores de ambos arreglos sean compatibles, con el propósito de facilitar la reestructuración dinámica de arreglos para el caso de los lenguajes que pudieran requerirla.

4.- Representación y Manejo de Subprogramas

En algunos lenguajes de programación modernos, los subprogramas son considerados como variables pertenecientes a una forma especial de tipos de datos estructurados (Tanenbaum, 1976). Estos tipos de datos están normalmente definidos por el número, posición y tipo de datos de sus parámetros formales y, en el caso de las funciones, además por el tipo de datos del valor que retornan. En estos lenguajes, el valor de una variable perteneciente a alguno de estos tipos de datos, es decir, el valor de un subprograma particular, habitualmente está definido por los nombres de sus parámetros formales y por el conjunto de sentencias que especifican su acción.

La máquina objeto virtual propuesta permite representar los subprogramas como variables pertenecientes a algún tipo de datos como los descritos. Para esto, el código de un subprograma, incluyendo el código necesario para asociar los parámetros formales con los correspondientes parámetros reales, es almacenado en el área de código de la memoria, una vez traducido al lenguaje de ensamble de la máquina. En el área de stacks, el subprograma es representado por la dirección de su descriptor. Un descriptor de subprograma (SD) es una estructura de bytes que contiene la información necesaria para permitir la ejecución del subprograma. Esta información consiste en la dirección de la primera instrucción del código correspondiente, el nivel de anidación estático con respecto al programa principal, la dirección base del registro de activación del subprograma en el cual está declarado, y una especificación del tipo de datos del valor que eventualmente retorna.

Normalmente, la interpretación de una llamada a un subprograma se traduce en el almacenamiento en el stack vigente de un nuevo registro de activación. La información básica contenida en estos registros permite hacer referencias a objetos que no son locales al subprograma que está siendo ejecutado, así como retornar el control al subprograma que hizo la llamada, una vez terminada tal ejecución. Esta información consiste en:

- La dirección de la instrucción que debe ser ejecutada inmediatamente después del retorno (dirección de retorno); corresponde al contenido del registro PC en el momento en que se produce la llamada;

- La dirección base del registro de activación en el que se produjo la llamada (enlace dinámico); corresponde al contenido del registro AR en el momento en que se produce la llamada;

- El nivel de anidación estático con respecto al programa principal; se obtiene del descriptor del subprograma;

- La dirección base del registro de activación más reciente en el cual el subprograma está declarado (enlace estático); también se obtiene del descriptor del subprograma.

En conjunto, la dirección de retorno y el enlace dinámico permiten retornar al subprograma que hizo la llamada. Por su parte, el nivel de anidación y el enlace estáticos permiten manejar las referencias a los objetos no locales al subprograma que está siendo ejecutado, en aquellos lenguajes en los que la asociación entre nombres y objetos es de tipo estática. Las instrucciones provistas para el manejo de los subprogramas son las siguientes:

- PushSD almacena en el stack vigente el descriptor de un subprograma; el nivel de anidación estático y la dirección base del código del subprograma deben ser especificados como operandos explícitos de la instrucción;

- SaveB reserva en el stack vigente el espacio necesario para almacenar un valor de tipo Byte; hay una instrucción similar para cada uno de los siete tipos de datos básicos de la máquina;

- CreateAR reserva en el stack vigente el espacio necesario para almacenar la información básica de un nuevo registro de activación;

- Call inicializa el contenido del registro de activación recién creado para comenzar la ejecución de un subprograma, con la información básica correspondiente;

- AssignB almacena en el registro de activación en el que se produjo la llamada el valor de tipo Byte retornado por el subprograma cuya ejecución está terminando; hay una instrucción similar para cada uno de los siete tipos de datos básicos de la máquina;

- Return reestablece, al terminar la ejecución de un subprograma, los contenidos que los registros SP, PC y AR tenían antes de iniciarse tal ejecución.

5.- Representación y Manejo de Múltiples Procesos

Los lenguajes de programación modernos de más reciente aparición, y otros no tan recientes, proveen facilidades para la definición y existencia simultánea de varios procesos activos. En general, estas facilidades no dependen de la posibilidad real de ejecutar en un computador dos o más procesos distintos en paralelo, sino que se basan en la existencia de interfaces que adecúan el número de procesos activos simultáneamente al número de procesadores disponibles. En el caso particular de la máquina propuesta, que consta de un único procesador, la estrategia consiste en asignar secuencialmente un intervalo igual de tiempo de ejecución a cada proceso activo.

Al iniciarse el intervalo de tiempo de ejecución asignado a un determinado proceso, la máquina obtiene la información necesaria para tal ejecución a partir del descriptor del proceso, almacenado en el área del heap. Un descriptor de proceso (PD) es una estructura de bytes que almacena el contenido que tenía cada uno de los cuatro registros del computador la última vez que se suspendió la ejecución del proceso correspondiente. Además, contiene las direcciones de los límites del área de stacks asignada al proceso y la dirección base del stack del proceso que hizo la activación. Durante la ejecución de un programa, existe un descriptor para cada uno de los procesos activos.

El tamaño máximo del stack asignado al proceso principal es definido por la máquina. El tamaño máximo del stack asignado a cada proceso activado durante la ejecución del proceso principal debe ser especificado en el instante de la activación. Para esto la máquina provee la instrucción CreateProcess que permite especificar en sus operandos explícitos dicho tamaño, así como de la dirección base del código correspondiente en el área central.

La activación de un proceso se traduce en la creación de un nuevo stack, que es almacenado en el área de stacks e inicializado con el registro de activación correspondiente al proceso. En estos registros, cuya estructura es idéntica a la de los subprogramas, el nivel de anidación es 0, la dirección de retorno y el enlace estático contienen ambos una dirección nula, y el enlace dinámico apunta al descriptor del proceso en el que se produjo la activación. En el caso particular del registro de activación del proceso inicial el enlace dinámico también contiene una dirección nula.

Debe notarse que no es posible asegurar la validez de los enlaces estáticos o dinámicos entre los procesos y los subprogramas en los cuales han sido declarados o activados. Esto se debe a que la ejecución de un subprograma puede terminar antes que la ejecución del proceso declarado o activado dentro de él, de modo que no puede asegurarse la existencia del registro de activación del subprograma durante todo el tiempo de ejecución del proceso: si durante la ejecución de un proceso se produce una referencia a un objeto no local, no puede asegurarse que ésta se pueda resolver siguiendo un enlace estático o dinámico al subprograma que contiene la declaración o produjo la activación del proceso, ya que este subprograma pudo haber terminado su ejecución antes de que se produjera la referencia.

De acuerdo con las características de los lenguajes que proveen procesos múltiples, el proceso (y no el subprograma) en el que se produce la activación de un segundo proceso, debe permanecer activo durante la ejecución de este último. Esto permite hacer referencias a objetos no locales al proceso que está siendo ejecutado, las que se resuelven siguiendo la cadena de enlaces dinámicos entre los registros de activación de los procesos y los descriptores de los procesos que hicieron las activaciones.

La búsqueda de un objeto en el stack de un proceso puede ser implementada de dos maneras diferentes. Por una parte, puede hacerse sólo entre las variables globales del proceso, para las que en todo instante existe una única instancia en el registro de activación inicial del stack correspondiente. Por otra, puede hacerse a partir del registro de activación que está en el tope del stack del proceso, y siguiendo la cadena de enlaces dinámicos entre los demás registros de activación. En este caso, se tiene acceso a todos los objetos activos en el stack, pero dos referencias a un mismo objeto pueden conducir a instancias diferentes de éste (ya que entre una referencia y otra puede producirse una nueva activación del subprograma al que pertenece el objeto), siendo responsabilidad del diseño del lenguaje correespondiente definir en forma adecuada el manejo de estas referencias.

Al finalizar la ejecución de un proceso, se libera el espacio ocupado por éste en el área de stacks y se elimina el descriptor correspondiente en el área del heap. Las eventuales referencias desde un proceso activo a objetos locales a un proceso cuya ejecución ya terminó detienen la ejecución del programa. La ejecución de un programa termina normalmente cuando ha terminado la ejecución de todos los procesos activados por el programa.

6.- Características del Lenguaje de Ensamble

Las instrucciones del lenguaje de ensamble de la máquina virtual son ejecutadas por un intérprete escrito en el lenguaje de programación C (Kernighan y Ritchie, 1978). La forma general de las instrucciones consiste en el nombre de la instrucción seguido por los operandos explícitos que puedan corresponder. La mayoría de las instrucciones, sin embargo, no tiene operandos explícitos. En estos casos, los operandos necesarios para la ejecución de las instrucciones se encuentran normalmente en el área de stacks o bien en el área del heap, y las instrucciones tienen acceso a ellos a través de los registros, ya sea directa o indirectamente (Hunter y Farquar, 1984).

En el caso de las instrucciones cuyos operandos, explícitos o implícitos, pertenecen a alguno de los siete tipos de datos básicos de la máquina, existe una versión de la instrucción para cada tipo de datos. Por ejemplo, el computador provee las instrucciones MultB, MultSW, MultDW, MultQW, MultSF, MultDF y MultQF para realizar la multiplicación entre dos valores de un mismo tipo de datos, estando ambos en el tope del stack vigente.

Similarmente, en el caso de las instrucciones cuyos operandos, explícitos o implícitos, representan direcciones de memoria, existe una versión de la instrucción para manejar direcciones absolutas y otra para manejar distancias relativas entre direcciones absolutas. Por ejemplo, el computador provee las instrucciones JumpAbs y JumpRel para transferir el control a una instrucción particular. El operando de la primera instrucción representa una dirección absoluta en el área de código de la memoria; el operando de la segunda instrucción representa una distancia, en bytes, con respecto a la dirección actual.

Las instrucciones provistas también facilitan la generación de código para las estructuras de control más comunes de los lenguajes de programación modernos. Por ejemplo, una sentencia del tipo

if A = B then sent1 else sent2,

puede ser fácilmente traducida a una secuencia de instrucciones tal como

```
        {instrucciones para almacenar los valores de A y B en el stack}
        BranchEQAbs   dir1
        JumpAbs       dir2
        ...
dir1:   {instrucciones correspondientes a sent1}
        BlockReturn
dir2:   {instrucciones correspondientes a sent2}
        BlockReturn
```

En este ejemplo, dir1 y dir2 son las direcciones en el área central de los códigos correspondiente a las instrucciones generadas para sent1 y sent2, respectivamente. Así, una vez que los valores de A y B han sido almacenados en el tope del stack vigente, la instrucción BranchEQAbs almacena el contenido actual del registro PC y, si ambos valores son iguales, lo reemplaza por la dirección dir1; en caso contrario, la instrucción JumpAbs almacena en el registro PC la dirección dir2. La secuencia de instrucciones correcta se ejecuta a continuación y luego la instrucción BlockReturn reestablece el contenido original del registro PC para continuar la ejecución del programa.

Además se incluyen instrucciones del tipo Branch para los diferentes tipos de comparaciones (LT, GT, NE, LE, GE), y las instrucciones BlockReturnT y BlockReturnF para terminar la ejecución de iteraciones de los tipos for, repeat y while.

La máquina virtual no provee instrucciones para realizar operaciones de entrada y/o salida de datos; provee, sin embargo, la instrucción ExecuteExternal que permite ejecutar rutinas definidas externamente al lenguaje de ensamble, como parte de la definición de un determinado lenguaje de alto nivel. El nombre de la rutina externa debe ser especificado como operando de la instrucción. Esto es, el código generado por el compilador de un lenguaje de alto nivel puede contener llamadas a estas rutinas, las cuales serán procesadas correctamente por el intérprete del lenguaje

de ensamble si han sido definidas apropiadamente. Estas definiciones pueden incluir el uso de otras instrucciones del lenguaje de ensamble, y deben informarse al intérprete mediante la inserción de los nombres de las rutinas externas en una tabla especial. La comunicación de parámetros entre las rutinas externas y el intérprete se hace a través del área del heap. Para esto, el computador provee la instrucción Lock, que impide el acceso al byte cuya dirección ha sido especificada como operando de esta instrucción.

Finalmente, el código generado por un compilador puede contener labels para identificar determinadas instrucciones, así como referencias a estas instrucciones hechas a través de los labels. El intérprete del lenguaje de máquina solucionará estas referencias en el momento de cargar el código en el área central de la memoria, traduciéndolas a direcciones absolutas o bien a distancias relativas entre direcciones absolutas.

7.- Conclusiones

Se ha descrito una máquina virtual que provee las herramientas básicas para representar las construcciones usuales presentes en la mayoría de los lenguajes de programación modernos. En especial, se ha puesto énfasis en facilitar la implementación de arreglos que pueden ser reestructurados dinámicamente durante la ejecución de un programa, en la representación de subprogramas como tipos de datos, y en el manejo de múltiples procesos en un único procesador. En este sentido, se ha propuesto un esquema para resolver las referencias a los objetos no locales a un determinado proceso. Además, los recursos descritos ofrecen la flexibilidad necesaria para incorporar variaciones; en particular, el conjunto de instrucciones del lenguaje de ensamble puede ser fácilmente extendido por el usuario del sistema.

AGRADECIMIENTOS

Este trabajo es parte del proyecto DIUC Nº 19/86 "Desarrollo de un Ambiente para la Generación de Compiladores", financiado por la Dirección de Investigación de la Universidad Católica de Chile, organismo al que los autores desean manifestar su agradecimeinto.

REFERENCIAS

1.- Campos, A., Eterovic, Y. "A Compiler Production Environment". Presentado para consideración a The Fifteenth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, San Diego, Enero, 1988.

2.- Eterovic, Y., Fuller, D. "Preliminary Design of a High Level Language Machine Architecture". Anales VI Congreso Chileno de Ingeniería Eléctrica, 469-474, Santiago, Chile, 1985.

3.- Ghezzi, C., Jazayeri, M. Programming Languages Concepts. John Wiley, New York, 1982.

4.- Hunter, C.B., Farquar, E. "Introduction to the NS16000 Architecture". IEEE Micro, 26-47, April 1984.

5.- Jensen, K., Wirth, N. Pascal User Manual and Report. Springer Verlag, New York, 1975.

6.- Kernighan, B.W., Ritchie, D.M. The C Programming Language. Prentice- Hall, Englewood Cliffs, 1978.

7.- Pratt, T.W. Programming Languages: Design and Implementation. Prentice-Hall, Englewood Cliffs, 1984.

8.- Tanenbaum, A.S. "A Tutorial on ALGOL 68". ACM Computing Surveys, 155-190, June 1976.

9.- Tanenbaum, A.S. "Implications of Structured Programming for Machine Architecture". Communications of the ACM, 237-246, March 1978.

10.- USDD. Reference Manual for the Ada Programming Language. U.S. Department of Defense, Washington, D.C., 1983.

11.- Wirth, N. Programming in Modula-2. Springer-Verlag, Berlin, 1985.

BASES PARA UN LENGUAJE DE DESCRIPCION DE LA SEMANTICA DE LOS LENGUAJES DE PROGRAMACION

A. Campos, Y. Eterovic, G. García, R. Santis
Dpto. de Ciencia de la Computación
P. Universidad Católica de Chile
Casilla 6177 - Correo 22
Santiago - CHILE

RESUMEN

Se presentan las bases para un lenguaje que permite describir la semántica de los lenguajes de programación. El esquema propuesto está orientado, principalmente, a facilitar la producción automática de analizadores semánticos, que puedan ser usados en compiladores para los lenguajes descritos con este mecanismo. Para estos efectos se considera que el proceso de generación de código intermedio es parte integrante de la fase de análisis semántico de los compiladores. El esquema de descripción resultante, que se basa en el empleo de sintaxis abstracta para su definición, es claramente operacional. Se da especial énfasis a facilitar la descripción de las características más comunes en los lenguajes de programación de mayor uso en la actualidad. El lenguaje de descripción provee herramientas para el control de flujo, la definición de dominios semánticos, el manejo de la tabla de símbolos, y la generación de código intermedio.

1.- Introducción

En el último tiempo, la construcción de compiladores se ha visto facilitada por la aparición de diversas herramientas computacionales, que permiten generar en forma automática los módulos que implementan algunas de sus fases. Esto ha permitido que la tarea de escribir un compilador completo se haya reducido a escribir la entrada a estos programas, y a escribir aquellas partes que no son generadas en forma automática.

Actualmente existen generadores para las fases de análisis léxico, como Lex (Lesk, 1975), y de análisis sintáctico, por ejemplo Yacc (Johnson, 1975). Estos generadores y sus similares son programas que, a partir de una descripción formal de las características léxicas y sintácticas del lenguaje de programación, producen un módulo para la fase respectiva de un compilador de ese lenguaje.

También se han construido algunos generadores automáticos de generadores de código, por ejemplo el descrito por Glanville y Graham (1978), que a partir de una especificación de las características de la máquina objeto producen un generador de código para esa máquina. Por otra parte, el proceso de producción de analizadores semánticos ha resultado más resistente a la automatización, aunque también se han hecho esfuerzos importantes en esa dirección (Sethi, 1983).

En este artículo se presentan las bases de un lenguaje para describir la semántica de los lenguajes de programación. El mecanismo propuesto está orientado, principalmente, a que estas descripciones permitan generar, en forma automática, analizadores semánticos para los compiladores de esos lenguajes. Con este objetivo práctico en mente, pierde relevancia la descripción de los conceptos y entes semánticos en sí misma, y adquiere mucho más importancia el tener mecanismos conceptualmente claros y de fácil aplicación a los lenguajes de programación. El esquema resultante para el lenguaje de descripción de la semántica presenta un enfoque claramente operacional.

La definición de este lenguaje es parte de un proyecto para desarrollar un ambiente computacional que proveerá herramientas que permitan automatizar la producción de compiladores (Campos y Eterovic, 1988). En especial, se desea que este sistema facilite la descripción de aquellas características más comunes entre los lenguajes más usados en la actualidad. El lenguaje ha sido definido para ser usado como el medio a través del cual se describirán, en este ambiente, las fases de análisis semántico y de generación de código intermedio de los compiladores.

2.- Análisis Semántico

Durante la fase de análisis semántico, el compilador debe recoger y almacenar toda la información necesaria sobre los objetos que emplea el programa. Esta información puede estar explícita o implícita en el código fuente, dependiendo del tipo de información y del lenguaje de programación. La estructura de datos usada para este fin se conoce con el nombre de tabla de símbolos. Sin embargo, ella no es necesariamente una tabla propiamente tal, puede ser cualquier estructura de datos apropiada, incluso una base de datos (Horwitz y Teitelbaum, 1986). La información almacenada en esta estructura es

usada posteriormente para realizar las verificaciones semánticas, y también para la generación de código.

Las verificaciones semánticas incluyen todas aquellas reglas del lenguaje que no pueden ser impuestas mediante la sintaxis. Su objetivo es detectar el uso de construcciones que a la luz de la definición del lenguaje de programación producen programas que, aunque gramaticalmente correctos, no tienen un significado definido. Importante entre ellas, es la revisión de la compatibilidad de tipo entre los objetos usados en el programa, y la consiguiente determinación de las coerciones que puedan ser necesarias.

Se ha estimado conveniente incluir en esta misma etapa la generación de código intermedio. Desde un punto de vista conceptual, el código a ser generado es, obviamente, una definición del significado del programa, es decir, de su semántica. Por otra parte, desde el punto de vista de la implementación, ambos procesos pueden definirse y realizarse en forma simultánea, evitando la necesidad de construir explícitamente una nueva representación intermedia del programa completo. El código intermedio generado por todos los compiladores producidos en este ambiente ha sido definido como el lenguaje de ensamble de una máquina de stack hipotética (Campos y otros, 1987). Esta máquina virtual sirve entonces de base para la definición de ciertos aspectos del lenguaje de descripción de la semántica.

3.- Sintaxis Abstracta

En el proceso de compilación, el análisis semántico se efectúa a continuación del análisis sintáctico. La entrada a esa fase consiste en una representación del programa que se está compilando, en la forma de un árbol sintáctico abstracto que es producido por el analizador sintáctico. Este árbol es una versión condensada del árbol de derivación que representa al proceso de análisis sintáctico (Aho y Ullman, 1977).

En este proceso previo, parte importante de los símbolos terminales presentes en el código fuente original pueden ser dejados fuera del árbol. En particular, todos aquéllos que proveen el azúcar sintáctica del lenguaje: la estructura del programa ya no está implícita en la secuencia lineal de símbolos usados en el programa fuente, sino que está explícitamente caracterizada por la estructura del árbol producido. Por otra parte, es usual que el analizador sintáctico haya resuelto también algunos problemas de índole semántica, por ejemplo aquéllos relativos a la asociatividad y prioridad de los operadores en una expresión, características que también quedan reflejadas en la estructura del árbol.

También es usual que al describir la sintaxis de un lenguaje de programación, y en particular al definir las categorías sintácticas, se tenga en consideración su semántica. Esto hace posible que una parte del trabajo

semántico pueda ser realizada a nivel sintáctico. Por todas estas razones resulta adecuado definir la semántica del lenguaje a partir de la definición de sus estructuras sintácticas. Sin embargo, no parece razonable comenzar exactamente desde el mismo punto, sino que es preferible comenzar desde algo mucho más simple: la sintaxis abstracta.

La sintaxis abstracta es una gramática libre de contexto que en sus producciones sólo contiene variables y símbolos terminales semánticamente significativos. Ella debe describir un lenguaje que incluya a aquél cuyos árboles de derivación son los árboles sintácticos abstractos producidos por el analizador sintáctico. Es decir, debe describir cada árbol que pueda ser producido durante el análisis sintáctico de un programa, pero puede describir, además, algunos árboles que nunca serán producidos debido a ciertas restricciones sintácticas del lenguaje de programación. Incluso, es perfectamente posible que sea una gramática ambigua, pues el árbol ya habrá sido construido durante el análisis sintáctico.

La sintaxis abstracta se emplea como base del lenguaje de descripción de la semántica. A su vez, el lenguaje empleado para definir las producciones de la sintaxis abstracta es muy simple y posee los siguientes metasímbolos:

=>	denota la producción.
%l(...)	denota una lista de lo que aparece entre paréntesis.
[...]	denota el nombre o número de una producción.

También es posible emplear comentarios que clarifiquen las producciones. Cualquier secuencia de carácteres que aparezca entre apóstrofes ('), o comillas ("), es considerada un comentario e ignorada como parte de la definición de la sintaxis abstracta. Así, por ejemplo, se puede definir:

```
[1]  A => 'program' %l(B) C
[2]  B => Identificador
[3]  C => 'begin' %l(Sentencia) 'end'
```

Esta definición indica que una lista de Identificador seguida de una lista de Sentencia es derivable desde A. Nótese que tanto "A", "B" y "C", como también "Sentencia" son variables de la gramática, y que "Identificador" representa, supuestamente, a un terminal en la definición sintáctica. Por otra parte, "program", "begin" y "end", que seguramente son símbolos terminales en la definición sintáctica, sólo aparecen como comentarios en esta definición, y su único objetivo es servir de ayuda mnemotécnica en su interpretación.

4.- Lenguaje de Descripción de la Semántica

La sintaxis abstracta sirve de punto de partida para describir la semántica de los lenguajes de programación. Para ello, a cada producción en

esta gramática se le asocian varios atributos. A su vez, estos atributos son empleados en la definición de otros atributos, asociados a otras o a la misma producción. Los atributos son, en realidad, rutinas o variables que representan acciones o valores, a través de los cuales se puede definir la transferencia de información entre los distintos nodos del árbol sintáctico abstracto. Mencionarlos en la definición de otro atributo equivale a especificar una llamada a la rutina, o el empleo del valor asociado a la variable.

Durante el análisis semántico, a través de las producciones, los atributos quedan asociados a los nodos correspondientes del árbol sintáctico abstracto del programa que se está compilando. El atributo que debe usarse en cada caso queda determinado en forma única por la estructura específica de este árbol. Cuando se requiere un atributo, ya sea una acción o una variable, para un nodo del árbol, se emplea la definición dada en la producción que corresponde a ese nodo. Es decir, aquélla en que la variable del lado izquierdo corresponde al nodo, y en que los símbolos del lado derecho corresponden exactamente a sus hijos.

El proceso que realiza el analizador semántico corresponde entonces a un recorrido sobre el árbol sintáctico abstracto, en que un mismo nodo podría ser visitado varias veces. Durante este recorrido, se realizan las acciones correspondientes a los atributos que sean requeridos por otros atributos. La forma exacta de este recorrido queda implícita en la descripción semántica al requerirse los distintos atributos definidos y, en general, será diferente para cada programa que se compile. Los atributos que son variables han sido introducidos, precisamente, como una herramienta que permite evitar que un mismo nodo sea visitado demasiadas veces. Ellos pueden almacenar valores calculados en una visita hasta cuando sean requeridos realmente, sin que sea necesario visitar una parte del árbol por segunda vez.

El lenguaje de descripción propiamente tal, que permite especificar los atributos, posee una serie de herramientas orientadas a facilitar la implementación de las características más comunes en los lenguajes de programación actuales. Estas herramientas no son rígidas, permitiendo que una misma tarea pueda ser definida de varias formas, con lo que se obtiene la ductilidad necesaria para acomodar las posibles variaciones. En particular, provee estructuras de control de flujo para una adecuada codificación de los atributos, permite la utilización de variables locales con el objetivo de disminuir el nivel de acoplamiento entre ellos, y permite la especificación y uso de otras rutinas y macros para facilitar su definición.

Por ejemplo, a una producción de la sintaxis abstracta para la suma de dos expresiones, se le podría definir el atributo "codigo" como sigue:

```
suma => exp '+' exp
    -codigo: begin
            exp_1.codigo
            exp_2.codigo
            Suma(exp_1.tipo, exp_2.tipo)
    end.
```

Este atributo representa una acción. Su evaluación se efectúa evaluando primero el atributo "codigo" de la primera expresión. A continuación se hace algo similar con la segunda expresión. Finalmente, se llama a la rutina o macro "Suma", pasando como parámetros los tipos de esas expresiones. Los tipos son también atributos de las producciones para las expresiones. Como ya se ha recorrido el sub-árbol correspondiente a cada expresión para evaluar el atributo "codigo", es posible que también se conozca el tipo resultante para cada una de ellas y que no sea necesario hacer un nuevo recorrido sobre esas partes del árbol.

Considerando que un objetivo muy importante de las descripciones semánticas hechas con este lenguaje es la generación automática de analizadores semánticos, también se han incluido herramientas para simplificar la definición de los dominios semánticos, el manejo de la tabla de símbolos, y la especificación del proceso de generación de código intermedio.

5.- Dominios Semánticos

Con el objetivo de formalizar los dominios semánticos, a la vez que clasificar los distintos entes que puedan aparecer en la descripción semántica de un lenguaje de programación, el lenguaje de definición dispone de algunos dominios predefinidos y provee mecanismos para la especificación formal de otros.

El lenguaje predefine los siguientes tipos de datos: INTEGER, REAL, BYTE, WORD y LABEL. Los cuatro primeros corresponden a tipos de datos presentes en prácticamente cualquier máquina real, y son equivalentes a algunos de los tipos básicos definidos en la máquina virtual cuyo código será generado. Por otra parte, el tipo de dato LABEL corresponde a nombres para direcciones de memoria, los que serán convertidos a la dirección absoluta real con posterioridad. Además, eventualmente, todos aquellos tipos de datos básicos que aparecen definidos en la máquina hipotética serán dominios predefinidos.

Para especificar nuevos dominios, a partir de aquéllos definidos previamente, el lenguaje provee las operaciones de unión y producto cartesiano, y también el concepto de función. Las definiciones pueden emplear, en forma recursiva, los dominios que están siendo definidos. Así, por

ejemplo, las siguientes son definiciones válidas para algunos tipos de datos posibles en un lenguaje de programación:

entero = INTEGER
doble = REAL × REAL
elemento = entero + REAL + doble
arreglo = INTEGER -> elemento
lista = (elemento × lista) + {λ}

En estas definiciones, el símbolo "+" representa la unión, "×" representa el producto cartesiano, y "->" representa la idea de función. El símbolo "λ" se usa para denominar a una lista en particular, que corresponde al concepto de la lista vacía o nula. Los símbolos "{" y "}" encierran un conjunto descrito en forma literal. Por último, el símbolo "=" representa la definición de un dominio, y los símbolos "(" y ")" se usan sólo para agrupar símbolos.

Estos mecanismos permiten definir los dominios básicos del lenguaje de programación cuya semántica se está describiendo. Por ejemplo, para Pascal se definirán los arreglos, los registros, etcétera. Al respecto, es importante destacar que no debe confundirse la definición de los dominios semánticos con la declaración, en un programa, de algunas instancias de él.

El objetivo principal de estas especificaciones es aclarar la relación conceptual entre los tipos de la máquina virtual, y los tipos propios del lenguaje que se describe. Con ellas, se obtiene también una manera precisa de describir las operaciones básicas de los tipos de este lenguaje; lo que permite, por ejemplo, definir las funciones de coerción, suma, o cualquier otra aplicable a un tipo de datos, o conjunto de ellos.

6.- Tabla de Símbolos

Se ha establecido un modelo general de tabla de símbolos de manera que sea fácil definir las funciones para administrarla en el caso de los lenguajes de programación de mayor uso en la actualidad, pero permitiendo también especificaciones diferentes. Para posibilitar la implementación de las distintas reglas de visibilidad de nombres presentes en los lenguajes de programación, la tabla de símbolos no existe como un todo único, sino que como un conjunto de tablas locales. Este conjunto se organiza como un grafo, usualmente un árbol, en que las tablas locales son los vértices. Este esquema permite definir en forma simple la regla de visibilidad de lenguajes como, por ejemplo, Pascal, en que un objeto se busca primero en una tabla local y, si no se encuentra allí, se busca en su ancestro dentro del árbol.

Para ciertos lenguajes de programación, el grafo puede simplificarse a un stack de tablas locales. Es por esto que en el lenguaje de descripción de la

semántica se proveen facilidades para definirlos y manipularlos. Específicamente, se predefinen las funciones para poner (Push) y sacar (Pop) una tabla local del stack, además de las funciones básicas para crear las tablas locales y los stacks. Concretamente, existen las siguientes operaciones predefinidas:

CreaTabla(t: Tabla)
CreaStack(s: Stack)
Push(t: Tabla, s: Stack)
Pop(s: Stack)

Los objetos almacenados en estas tablas locales se denominan símbolos. En general, ellos representan a los diferentes entes usados en un programa sobre los cuales se desea mantener alguna información. En este modelo, un símbolo consta de un nombre que lo identifica y de una serie de atributos. A su vez, los atributos son tríos compuestos por su nombre, tipo y valor. El nombre de cada atributo es único para un símbolo dado, y sirve para identificarlo. Los nombres del símbolo y de los atributos son valores de tipo IDENTIFICADOR. El valor de un atributo puede ser un valor numérico, una secuencia de carácteres, o puede ser un puntero a otro símbolo, a una tabla local, o a un trozo de código.

El lenguaje provee funciones primitivas que permiten crear y buscar un símbolo en una tabla local, agregar y eliminar un atributo a un símbolo, y modificar el valor de un atributo:

CreaSimbolo(n: IDENTIFICADOR, t: Tabla): Simbolo
BuscaSimbolo(n: IDENTIFICADOR, t: Tabla): Simbolo
AgregaAtributo(s: Simbolo, n: IDENTIFICADOR, t: Tipos, v: Tipo)
BorraAtributo(s: Simbolo, n: IDENTIFICADOR)
ModificaAtributo(s: Simbolo, n: IDENTIFICADOR, v: Tipo)

Como herramienta para facilitar el uso de los stacks de tablas locales, también se predefine una función que busca un símbolo dentro de un stack:

BuscaEnStack(n: IDENTIFICADOR, s: Stack): Simbolo

Dado un nombre y el stack de tablas, esta función busca un símbolo con ese nombre. La búsqueda se hace en cada tabla local, comenzando desde la que está en el tope del stack, y siguiendo ordenadamente hacia abajo. Devuelve el primer símbolo que encuentra con ese nombre, o un símbolo nulo en caso de que no existiera uno.

Si se quisiera implementar otro tipo de búsqueda, o para tablas de símbolos en que las tablas locales están arregladas de otra forma, se deberá programar la función correspondiente. Para ello se puede emplear como base

la función de búsqueda en una tabla local (BuscaSimbolo) y un número de punteros a otras tablas.

Otra facilidad provista en el lenguaje de descripción semántica es una función generadora de símbolos internos. Esta función crea nombres únicos para símbolos, garantizando que será diferente al de cualquier otro símbolo generado por ella, o al nombre de cualquier objeto que aparezca en el programa. Esto permite incluir símbolos interno creados por el analizador semántico y que no provienen, necesariamente, de objetos del programa.

7.- Generación de Código Intermedio

Como se dijo anteriormente, a cada producción de la sintaxis abstracta se le definen atributos que, durante el análisis semántico, quedan asociados al nodo correspondiente en el árbol sintáctico abstracto. Estos atributos pueden ser rutinas que describen acciones, y cuyos parámetros son los hijos del nodo respectivo. El paso de estos parámetros queda implícito en la definición del atributo.

Este mismo mecanismo es el que debe usarse para especificar el proceso de generación de código intermedio. Si se requiere la generación de ciertas instrucciones, debe haber un atributo que lo haga. El lenguaje de descripción semántica provee, además, funciones que permiten emitir las instrucciones de la máquina virtual a un archivo, desde donde pueden ser posteriormente optimizadas, interpretadas o traducidas a código de una máquina objeto real. En caso que se usara otro conjunto de instrucciones como lenguaje intermedio para los compiladores, estas rutinas pueden redefinirse con facilidad.

En el ejemplo del atributo "codigo", definido en la producción para la suma de dos expresiones y presentado anteriormente, se puede suponer que él genera las instrucciones necesarias para evaluar la suma. En ese caso, el atributo "codigo" de las expresiones debe emitir las instrucciones para dejar su valor en el tope del stack de la máquina hipotética. Esto puede hacerlo directamente, o a través de atributos definidos para las variables que aparezcan al lado derecho de producciones para las expresiones. Con posterioridad, la rutina o macro "Suma" emitirá, de acuerdo al tipo de ellos, las instrucciones para sumar dos valores que están en el tope del stack.

Es importante destacar que esa definición es independiente de si se genera código o no. Por ejemplo, también se podría interpretar como una especificación del valor de una suma. En ese caso, el atributo "codigo" representaría el proceso de evaluación de una suma o de una expresión. Este proceso dejaría los valores calculados en el tope de un stack de evaluación, y la rutina o macro "Suma" tomaría esos valores y produciría el resultado final.

8.- Conclusiones

Se han presentado las bases para la definición de un lenguaje que permite describir la semántica de los lenguajes de programación, cuyo objetivo principal es el de facilitar la producción de un analizador semántico a partir de esas descripciones. El resultado es un lenguaje de corte netamente operacional, que incluye algunas facilidades para definir con simplicidad las características más usuales entre los lenguajes de mayor difusión en la actualidad. Sin embargo, también posee la flexibilidad suficiente para acomodar alternativas posibles de presentarse en los lenguajes de programación. Se ha dado especial importancia a características que permitan definir con comodidad los tipos involucrados, las acciones a efectuar por el administrador de la tabla de símbolos, y el proceso de generación de código intermedio.

AGRADECIMIENTOS

Este trabajo es parte del proyecto DIUC 19/86: "Desarrollo de un Ambiente para la Generación de Compiladores", que ha sido financiado por la Dirección de Investigación de la Pontificia Universidad Católica de Chile, organismo al cual los autores desean expresar sus más sinceros agradecimientos.

REFERENCIAS

1.- Aho, A.V. y Ullman, J.D. (1977) Principles of Compiler Design. Addison-Wesley, Reading.

2.- Campos, A., Eterovic, Y. (1988) A Compiler Production Environment. Presentado para consideración a The Fifteenth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, San Diego, Enero 1988.

3.- Campos, A., Eterovic, Y., García, G., Santis, R. (1987) Implementación de un Computador Virtual con Capacidad de Multiprocesamiento. Aceptado en el Octavo Congreso Latinoamericano e Ibérico sobre Métodos Computacionales para Ingeniería. Rio de Janeiro, Noviembre 1987.

4.- Glanville, R.S. y Graham, S.L. (1978) A New Method For Compiler Code Generation. Proceedings of the Fifth Annual ACM Symposium on Principles of Programming Languages, New York, Enero 1978, 231-240.

5.- Horwitz S. y Teitelbaum, T. (1986) Generating Editing Environments Based on Relations and Attributes. ACM Transactions on Programming Languages and Systems, Vol 8, Nº4, 577-608.

6.- Johnson, S.C. (1975) Yacc: Yet Another Compiler-Compiler. CSTR 32, Bell Laboratories, Murray Hill.

7.- Lesk, M.E. (1975) LEX - A Lexical Analyzer Generator. CSTR 39, Bell Laboratories, Murray Hill.

8.- Sethi, R. (1983) Control Flow Aspects of Semantic-Directed Compiling. ACM Transactions on Programming Languages and Systems, Vol 5, Nº4, 554-595.

Desarrollo e implementación de un interpretador de lenguaje Prolog portable y extendible

Autores: Ing. Luis Roberto Ojeda Ch.
Ing. Camilo Parra

Dirección: Departamento de Ingeniería de Sistemas
Universidad Nacional de Colombia. Bogotá

Una labor en la apertura del Laboratorio de Inteligencia Artificial de la Universidad Nacional de Colombia será la implementación de una herramienta de desarrollo en el área de IA y Sistemas Expertos con los siguientes objetivos:

- Lograr el acceso a grandes bases de datos o de conocimientos construídos en ambientes y lenguajes diferentes.
- Facilitar el uso del interpretador -por su portabilidad- en diferentes sistemas operativos y en diferentes computadores, desde el orden de microcomputadores hasta mainframes, con la única condición de tener un compilador en lenguaje C.
- Facilitar la extensión, para ampliar sus posibilidades en el desarrollo de sistemas expertos con interfases gráficas y para lograr interfases con lenguajes como SQL u otros lenguajes manejadores de Bases de Datos.
- Facilitar la investigación y desarrollo en el área de programación lógica con el desarrollo e implementación de nuevos predicados en el lenguaje Prolog.
- Ser soporte para investigación en áreas afines como el de Bases de Datos Estadísticos.

Agradecemos las inquietudes observaciones y aportes que se puedan presentar en el ambiente universitario y científico, El resultado del proyecto estará a disposición de las instituciones latinoamericanas que tengan interés en el área.

INGENIERIA DE SOFTWARE

TECNICAS DE PROGRAMACION

VERIFICAÇÃO E VALIDAÇÃO DE SOFTWARE NA FASE DE ESPECIFICAÇÃO DE REQUISITOS

Clifton Eduardo Clunie Beaufond
Universidad Tecnológica de Panamá
Faculdad de Ingenieria en Sistemas Computacionales
Aptdo. 6-2894 - ElDorado
Panamá - Rep. de Panamá

Ana Regina Cavalcanti da Rocha
Universidade Federal do Rio de Janeiro - COPPE
Caixa Postal 68511
21945 - Rio de Janeiro - RJ - Brasil

RESUMO

A qualidade de um sistema de software, como a de qualquer produto de alta tecnologia, depende da qualidade de suas especificações. As especificações são a base do desenvolvimento futuro do software e constituem um instrumento de predição e avaliação do comportamento do software a ser construído.A falta de especificações adequadas é uma das principais causas da produção de software de baixa qualidade e com custos de desenvolvimento, operação e manutenção elevados.

Neste artigo, examina-se o problema da qualidade das especificações. A partir de um modelo de avaliação de qualidade de especificações, são identificadas características primitivas e definidos processos de avaliação. Desta forma, uma especificação pode ser avaliada numa forma organizada e disciplinada.

INTRODUÇÃO

Qualidade é um atributo associado a alguma coisa. Assim sendo, qualidade não pode ser definida universalmente mas deve ser definida para o item em questão. (qualidade de especificações, qualidade de metodologias, qualidade de programas, etc). Qualidade é, também, um conceito multidimensional que se realiza através de um conjunto de atributos ou características.

A experiência têm demonstrado que a qualidade do software é uma responsabilidade de todos os que participam na sua realização ou em seu uso. A qualidade deve ser construída em cada fase do ciclo de desenvolvimento. Não se pode adicionar qualidade em etapas posteriores e por pessoal que não participou do desenvolvimento.

Para que uma organização possa construir software com a qualidade desejada e se possa medir se o software produto possui esta qualidade, é necessário que se defina a qualidade de software de acordo com seu contexto. Para isto deve-se analisar a natureza do produto e identificar-se quais são as características consideradas como desejáveis. Estas características devem ser possíveis de serem avaliadas, existindo um grau mínimo a ser atingido para que um software seja considerado de boa qualidade.

Cabe salientar que a qualidade do software depende da satisfação do usuário com o serviço fornecido pelo software e com a facilidade de seu uso. Mas existem diversos tipos de usuários (operador, mantenedor, auditor, etc) e a qualidade obtida depende da satisfação das necessidades e expectativas de todos estes usuários. Podemos definir, então, qualidade de software como o conjunto de propriedades a serem satisfeitas em determinado grau, de modo a satisfazer as necessidades de seus usuários.

BOEHM (1) elaborou a curva da proporção entre custos de hardware e software, mostrando que a atividade de manutenção é uma proporção dos custos de software que aumenta progressivamente, tendo em vista o acúmulo de novos softwares produzidos e o aumento do volume de manutenção dos existentes. Mostrou, também, que o custo de correção de um erro na fase de operação de um sistema, isto é, durante a manutenção, é muitas vezes superior

ao custo de correção desse mesmo erro nas fases inciais do desenvolvimento. Percebe-se claramente a vantagem que advêm de detetar e corrigir cedo os erros.

Torna-se evidente, portanto, que o aumento de investimento nas fases iniciais do processo de desenvolvimento pode diminuir consideravelmente os gastos com manutenção, parcela maior do custo total do sistema.

A primeira fase do processo de desenvolvimento é a que, provavelmente, leva a mais erros, dada sua natureza subjetiva. Esta fase, denominada aqui especificação de requisitos, é onde se centra nosso trabalho. É altamente importante e vantajoso investir em esforços humanos e financeiros para gerar especificações de boa qualidade, se possível totalmente isentas de falhas. A qualidade dos requisitos definidos no início do processo de desenvolvimento é fator básico para que possa ser produzido um software de boa qualidade.

O PROBLEMA DE MEDIÇÕES

A determinação da qualidade é um fator importante no nosso viver diário. Convivemos com isso. A qualidade é avaliada de uma forma quase direta, por comparação de objetos sob idênticas condições, considerando-se atributos pré-determinados. Os objetivos são julgados de acordo com características tais como cor, odor, sabor, beleza, habilidade, etc. Entretanto, este tipo de julgamento é muito subjetivo e para ter algum valor é necessário que seja conduzido por especialistas.

A subjetividade e a experiência também são aplicadas à determinação da qualidade do software. Para ajudar a resolver o problema precisamos de um método para avaliação da qualidade de software através do qual possamos realizar medições quantitativas baseadas em uma análise objetiva.

Medição de software é uma área destinada à consignação de índices quantitativos do software em desenvolvimento, PERLIS (2). Tem como objetivo ajudar a responder às seguintes perguntas que vão surgindo na medida em que se vai progredindo no desenvolvimento:

- é o momento adequado para continuar com a etapa seguinte?
- será necessário retroceder a uma etapa anterior?

As respostas a estas perguntas devem possuir um certo grau de exatidão para que se possa continuar com o controle efetivo do processo de desenvolvimento.

DENICOFF e GRAFTON (3), mostram a necessidade de que a engenharia de software, esteja marcada pela disponibilidade de parâmetros de medidas, precisas e bem entendidas. No entanto medidas para software não podem ser, como nas demais disciplinas de engenharia, baseadas em ciências físicas. Isto torna necessário que a abordagem dada às medidas de software seja marcada pelo paradigma científico tradicional de hipótese, avaliação, crítica e revisão.

O principal objetico do estudo das medidas do software não é só a determinação estática das propriedades do software,mas, sim uma predição científica durante o ciclo de vida.

Devido a seu potencial preditivo, medidas podem ser utilizadas pelo menos de três formas, CURTIS (4):

- como ferramenta de informação gerencial, ajudando na predição de resultados futuros e consequências na área de custos, produtividade, duração do projeto, etc. Desta forma, oferecem indicadores confiáveis que ajudam a ter boa visão gerencial do projeto;
- como medida da qualidade de software, pois fornecem critérios quantificáveis sob os quais pode-se avaliar o software produto;
- como "feedback" para o pessoal envolvido no projeto.

Entretanto, medidas para serem úteis, precisam estar organizadas. A ciência moderna considera que a complexidade da realidade pode ser entendida pelo estabelecimento e compreensão de modelos hierárquicos, alguns descrevendo comportamentos macroscópicos, ignorando os detalhes e outros descrevendo um comportamento mais microscópico. A ciência trabalha,então, baseada na utilização de modelos, que explicam a relação entre o significado das variáveis sob estudo.

Toda experiência baseada num desenvolvimento científico cumpre uma sequência de etapas:

- início das observações (medições);
- formulação de hipóteses, baseadas num conjunto de axiomas, para explicar as observações;
- construção do modelo com suas variáveis, baseadas ou estimadas a partir de observações;
- utilização do modelo para fazer predições das propriedades do sistema;
- execução do experimento, (controlando as observações e o meio ambiente) para determinar a precisão do modelo.

A figura I mostra o ciclo de desenvolvimento de um modelo.

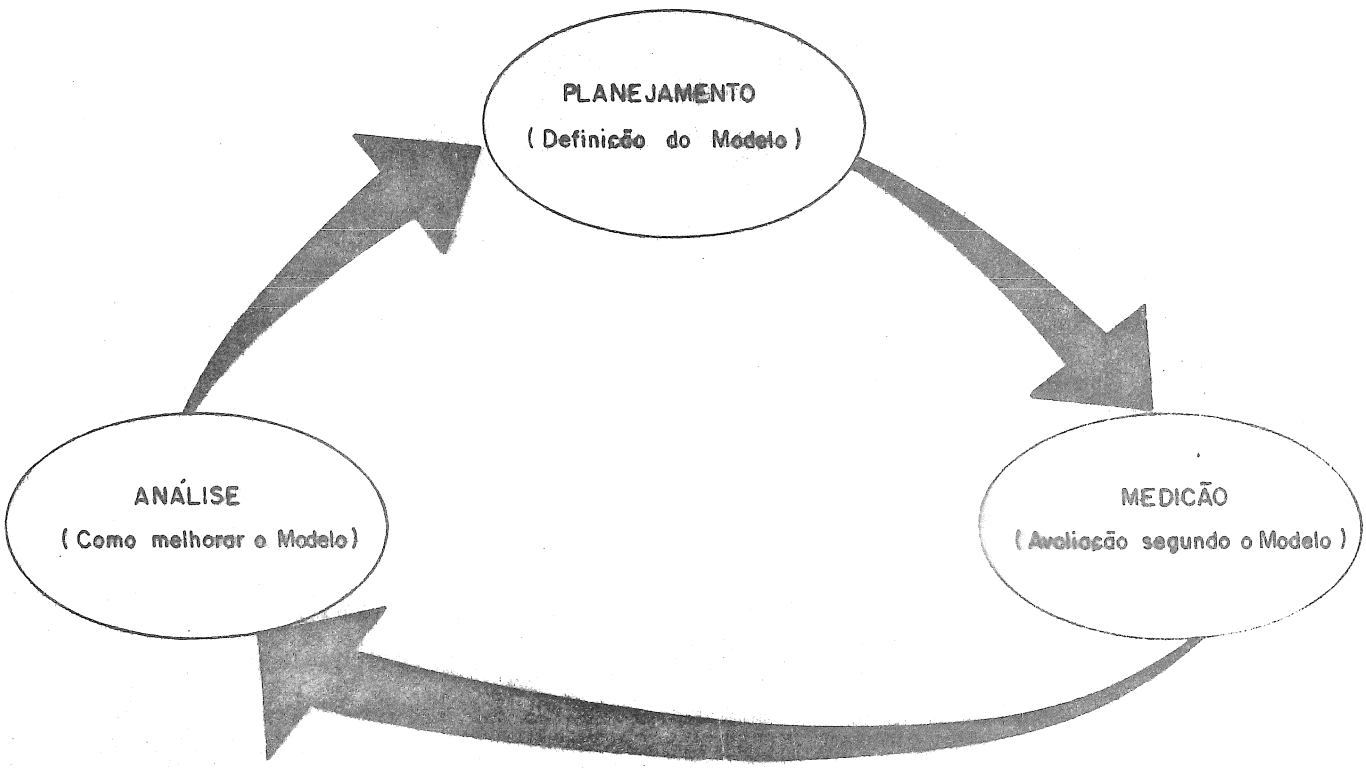


Figura I - Ciclo de Definição de Uso de um Modelo para Avaliação da Qualidade.

UM MODELO PARA AVALIAÇÃO DA QUALIDADE DE SOFTWARE

Baseada nas considerações anteriores e nos trabalhos de BOEHM (5) e McCALL (6) foi definido um Modelo para Avaliação da Qualidade de Software, baseado nos seguintes conceitos, ROCHA (7)

- Objetivos de qualidade: são propriedades gerais que o produto deve possuir;
- Fatores de qualidade: determinam a qualidade do ponto de vista dos diferentes usuários do produto (usuário final, mantenedores, etc);
- Critérios: são atributos primitivos possíveis de serem avaliados;
- Processos de avaliação: determinam o processo e os instrumentos a serem usados de forma a se medir o grau de presença,no produto, de um determinado atributo;
- Medidas: indicam o grau de presença de um determinado critério;
- Medidas agregadas: indicam o grau de presença de um determinado fator.

A figura II mostra a estrutura´ do Modelo.

Os objetivos de qualidade são atingidos através dos fatores de qualidade, que podem ser compostos por outros fatores e são avaliados através de critérios. Os critérios definem atributos de qualidade para os fatores.

Medidas são valores resultantes da avaliação de um produto segundo um critério específico.

Objetivos e fatores não são diretamente mensuráveis e só podem ser avaliados através de critérios. Um critério é um atributo primitivo, isto é, um atributo independente de todos os outros atributos. Nenhum critério isolado é uma descrição completa de um determinado fator ou sub-fator. Da mesma forma, nenhum fator define completamente um objetivo.

Numa primeira etapa esse modelo foi usado para definir objetivos, fatores e sub-fatores para os diferentes produtos gerados ao longo do desenvolvimento, ROCHA (9), ROCHA (10). Agora este modelo foi refinado definindo-se critérios e processos de avaliação para cada sub-fator.

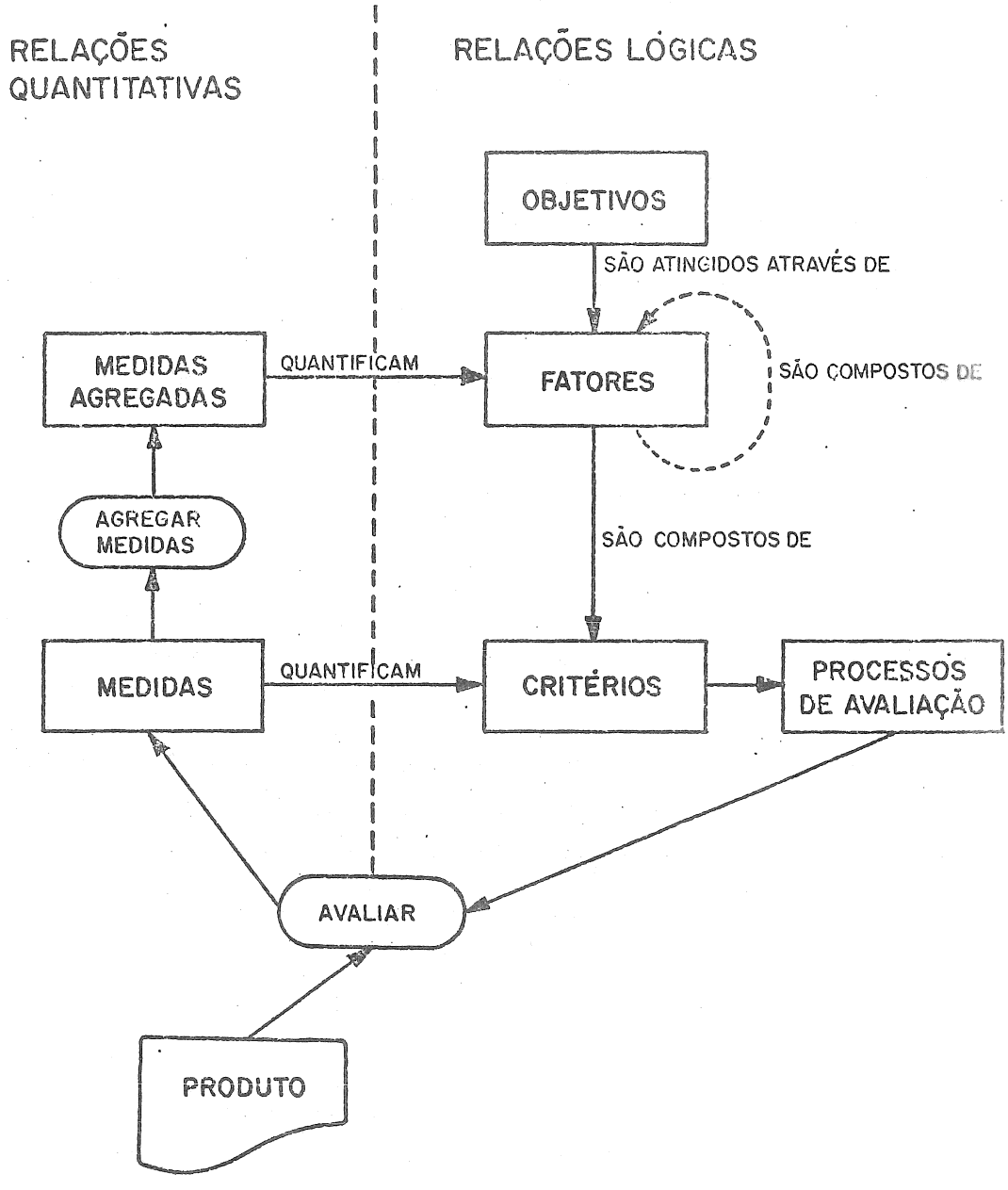


Figura II - Modelo para Avaliação da Qualidade de Software.
Fonte: ROCHA (8).

VERIFICAÇÃO E VALIDAÇÃO DE ESPECIFICAÇÕES

Ao desenvolver um software deve estar presente sempre a preocupação com a qualidade ao longo do processo de desenvolvimento. Portanto, é necessário que este processo de desenvolvimento se dê de forma segura e confiável. Para que isto aconteça, é necessário que os produtos intermediários (especificações, plano, projeto, etc) sejam avaliados quanto à sua qualidade, por esta implicar diretamente na qualidade do produto final. Isto evidencia a necessidade de duas atividades: verificação e validação.

BOEHM (11) define os dois conceitos, verificação e validação, de uma forma simples:

- por verificação entende-se "estamos construindo certo o produto"?;
- por validação entende-se "estamos construindo o produto certo"?;

Podemos então afirmar que:

- ao verificarmos um produto, estamos preocupados com sua qualidade intrínseca e com a qualidade de seu processo de construção, e,
- ao validarmos um produto, estamos preocupados com seu conteúdo.

No que se refere a especificações, foram identificados três objetivos de qualidade: confiabilidade da representação, confiabilidade conceitual e utilizabilidade, portanto:

- ao avaliarmos a confiabilidade da representação, estamos realizando uma verificação na especificação e;
- ao avaliarmos a confiabilidade conceitual estamos validando a especificação.

Para que seja possível avaliar se a especificação reflete o que usuário deseja, é necessário que ele seja capaz de ler e compreender a especificação. Além disso, as especificações são documentos lidos e manipulados por diversas pessoas não necessariamente conhecidas entre sí. Isso evidencia a importância da confiabilidade da representação.

O objetivo confiabilidade da representação é atingido através dos seguintes fatores de qualidade:

comunicabilidade: refere-se à capacidade de comunicação da especificação;
manipulabilidade: refere-se à facilidade de manipulação da especificação no seu uso, modificação, etc.

O fator comunicabilidade é atingido através dos seguintes sub-fatores de qualidade:

correção no uso da linguagem de especificação: é a característica de uma especificação de requisitos estar correta do ponto de vista do uso da linguagem de especificação;
concisão: é a característica de uma especificação de requisitos estar escrita com um mínimo de volume de texto, isto é, com uma maximização da quantidade de informações por unidade de texto;
uniformidade de terminologia: é a característica de uma especificação de requisitos estar escrita com uniformidade de terminologia e simbologia;
uniformidade de abstração: é a característica de uma especificação de requisitos estar escrita com o mesmo grau de detalhamento nos diversos níveis de abstração;
modularidade: é a característica de uma especificação de requisitos estar dividida em módulos, permitindo um perfeito entendimento de seus diveros aspectos de forma exclusiva.

O fator manipulabilidade é atingido através dos seguintes sub-fatores de qualidade:

disponibilidade: é a característica de uma especificação de requisitos estar disponível para o acesso de seus usuários, na versão mais atualizada.
estrutura: é a característica de uma especificação de requisitos estar organizada em seus documentos de forma a facilitar ao leitor a encontrar a visão que deseja;
rastreabilidade: é a característica de uma especificação de requisitos permitir o acompanhamento de uma sequência de agregação de detalhes relacionados a um determinado aspecto ou assunto.

O objetivo fundamental é construir um produto de acordo com as necessidades e expectativas do usuário. Isto evidencia a importância da confiabilidade conceitual. O objetivo confiabilidade conceitual é atingido através dos seguintes fatores de qualidade:

fidedignidade: refere-se à confiança de que a especificação representa, o que é entendido como sendo as necessidades e expectativas de seus usuários.
suficiência: refere-se à completeza, necessidade e maneira explícita com que os requisitos são especificados;

O fator fidedignidade é atingido através dos seguintes sub-fatores de qualidade:

fidelidade: é a característica de uma especificação de requisitos estar escrita coerentemente com as necessidades do usuário;
consistência: é a característica de uma especificação de requisitos estar escrita isenta de contradições;
não ambiguidade: é a característica de uma especificação de requisitos estar escrita isenta da possibilidade de diferentes interpretações para qualquer requisito.

O fator suficiência é atingido através dos seguintes sub-fatores de qualidade:

completeza: é a característica de uma especificação de requisitos conter todos os requisitos determinantes do software em desenvolvimento, completamente definidos;
necessidade: é a característica de uma especificação de requisitos estar escrita isenta de descrições inúteis;
ser explícita: é a característica de uma especificação de requisitos estar escrita isenta de definições implícitas.

Uma especificação deve poder ser utilizada para:
- realização de alterações;
- ser reutilizada em forma parcial ou total, em outros produtos similares;
- ser avaliada;
- ser implementada, depois de feito um estudo completo de sua viabilidade.

O objetivo utilizabilidade se realiza, portanto, através dos seguintes fatores de qualidade:

manutenibilidade: refere-se à capacidade de uma especificação permitir a introdução de alterações após ter sido inicialmente definida, construída e aceita;
reutilizabilidade: refere-se à capacidade de uma especificação ter suas partes organizadas e completamente desenvolvidas de maneira a permitir sua reutilização parcial ou total em outras aplicações;
avaliabilidade: refere-se à capacidade de uma especificação poder ser avaliada com relação a suas propriedades inerentes e com relação ao produto que especifica;
viablidade de implementação: refere-se à capacidade de uma especificação de requisitos poder ser implementada.

O fator manutenibilidade é atingido através dos seguintes sub-fatores:

evolutibilidade: é a característica de uma especificação de requisitos permitir refinamentos sucessivos;
modificabilidade: é a característica de uma especificação de requisitos permitir que alterações sejam feitas com facilidade, de forma completa e consistente, sem perda de qualidade.

O fator reutilizabilidade é atingido através dos seguintes sub-fatores:

aplicabilidade: é a característica de uma especificação de requisitos poder ser reutilizada ao se especificar outro produto na mesma área de aplicação;
adaptabilidade: é a característica de uma especificação de requisitos poder ser adaptada para corresponder, perfeitamente, a outra aplicação.

O fator avaliabilidade é atingido através dos seguintes sub-fatores:

verificabilidade: é a característica de uma especificação de requisitos poder ser avaliada em relação às propriedades desejáveis para um documento deste tipo;
validabilidade: é a característica de uma especificação de requisitos poder ser avaliada em relação ao produto que especifica.

O fator viabilidade de implementação é atingido através dos seguintes sub-fatores:

viabilidade econômica: é o atributo de se poder construir o software tendo como critério de avaliação a relação entre os custos e os benefícios;
viabilidade financeira: é o atributo de se poder construir o software tendo como critério de avaliação a existência e a disponiblidade de capital necessário para conduzir o desenvolvimento;
viabilidade tecnológica: é o atributo de se poder construir o software tendo como critério de avaliação a posse e o domínio da tecnologia necessária para conduzir o desenvolvimento;
viabilidade de mão-de-obra: é o atributo de se poder construir o software tendo como critério de avaliação a existência e a disponiblidade de mão-de-obra necessária para desenvolver o software;
viabilidade de cronograma: é o atributo de se poder construir o software tendo como critério de avaliação as restrições de cronograma;
viabilidade social: é o atributo de se poder construir o software tendo como critério de avaliação as implicações deste software na sociedade.

A figura III dá uma visão geral dos objetivos, fatores, sub-fatores e critérios que determinam a qualidade de especificações.

Para que seja possível realizar avaliações da qualidade de especificações foi elaborado um "Manual de Controle da Qualidade de Especificações" descrevendo detalhadamente objetivos, fatores, sub-fatores e processos de avaliação. Os anexos exemplificam o conteúdo deste Manual. O texto completo pode ser encontrado em CLUNIE (12).

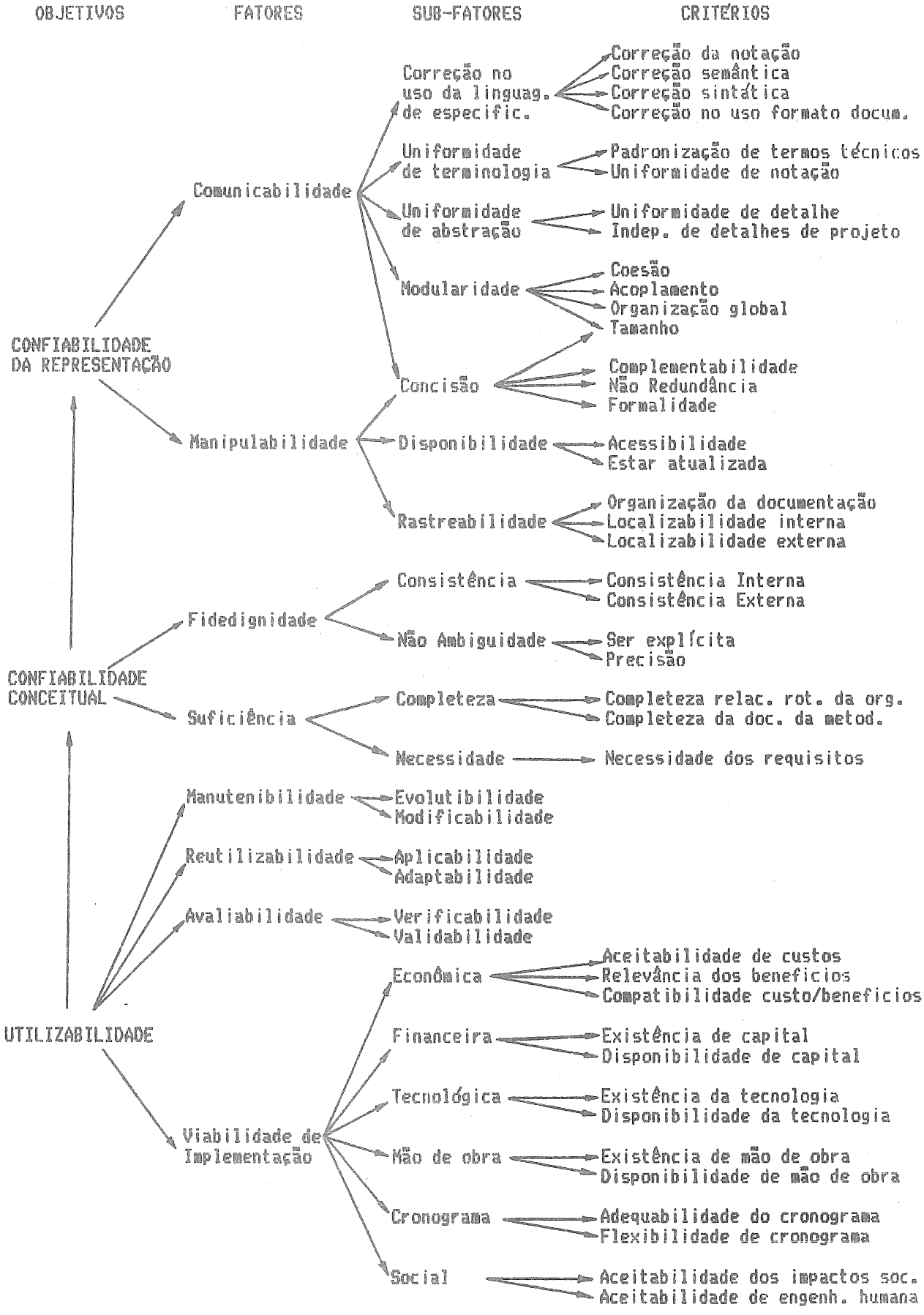


Figura III - Caracacterísticas de Qualidade da Especificação de Requisitos.

CONCLUSÃO

Neste trabalho partimos de um modelo para avaliação da qualidade de software e identificamos um conjunto de características primitivas que auxiliam na construção de especificações mais confiáveis. A partir destas características definimos um "Manual para Controle da Qualidade de Especificações", cujo objetivo é permitir avaliar uma especificação de forma organizada e disciplinada. A próxima etapa deste trabalho é a construção de um sistema especialista para avaliação de especificações.

REFERÊNCIAS

(1) Boehm, B. Software Engineering Economics New Jersey, Prentice Hall, 1981.

(2) Perlis, A.J. "Controlling Software Development Through the Life Cicle Model", in Software Metrics: An Analysis and Evaluation, MIT Press, 1981.

(3) Denicoff, M.; Graftan, R. "Software Metrics: A Research Iniciative", in Software Metrics: An Analysis and Evaluation MIT Press, 1981.

(4) Curtis, B. "The Measurement of Software Quality and Complexity", in Software Metrics: An Analysis and Evaluation, MIT Press, 1981.

(5) Boehm, B. et alli, Characteristics of Software Quality, Amsterdan, North-Holland, 1978.

(6) McCall, J. "An Introduction to Software Quality Metrics", in, Software Quality Management, Petrocelli, 1979.

(7) Rocha, A.R.C. da, Um Modelo para Avaliação de Qualidade de Especificações, Tese de Doutorado, PUC-RJ, junho de 1983.

(8) Rocha, A.R.C. da, Staa, A.V. "How to Improve Objectivity when Measuring Specifications", in, Fifth International Conference of the Israel Society for Quality Assurance, Tel Aviv, Israel, Outubro 1984.

(9) Rocha, A.R.C. da; Freitas, A.C.T., "How to Evaluate the Conceptual Reliability of Software Specifications", in, Third International Workshop on Software Specification and Design, Londres, UK, 1985.

(10) Rocha, A.R.C. da; Bargut, M.F.; "Quality Attributes for Microcomputer Programs", in, Sixth International Conference of the Israel Society for Quality Assurance, Tel Aviv, Israel, novembro de 1986.

(11) Boehm, B., "Verifying and Validating Software Requirements and Design Specification", IEEE Software, vol. 1, nº 1, janeiro de 1984.

(12) Clunie Beaufond, C.E. Verificação e Validação de Software na Fase de Especificação de Requisitos, Tese de Mestrado, COPPE/UFRJ, fevereiro de 1987.

Critério: Correção semântica

Objetivo: Confiabilidade da representação
Fator: Comunicabilidade
Sub-fator: Correção no uso da ling. de especific.

Definição:

A especificação possui o critério **correção semântica** na medida em que está escrita utilizando, de forma correta, o conjunto de regras semânticas pré-definidas na linguagem de especificação.

Discussão:

A sintaxe trata da maneira de reunir os signos para formar sentenças corretas em uma determinada linguagem. A semântica trata do significado contido nestas sentenças sintaticamente corretas. Embora seja possível gerar sentenças em uma linguagem obedecendo à regras sintáticas, não é garantido que estas sentenças tenham algum significado.

Regras sintáticas não são suficientes para caracterizar uma linguagem. É necessário introduzir-se um outro grupo de regras chamado de regras semânticas. A razão é óbvia: intérpretes podem partilhar a mesma estrutura lingufstica com regras semânticas diferentes sendo, então, incapazes de se comunicar.

É neste contexto que se introduz a dimensão semântica, a qual pode ser definida como o estudo da relação entre objetos e sua representação.

Processo de avaliação:

Este critério pode ser avaliado através da resposta à seguinte pergunta:

	SIM/NÃO 1 ou 0
1. a especificação está escrita obedecendo às regras semânticas pré-definidas na linguagem de especificação?	__

Interpretação dos resultados:

Rever o uso das regras semânticas, caso a resposta seja NÃO.

Atributos de qualidade relacionados:

Manutenibilidade: Especificações escritas obedecendo as regras semânticas pré-definidas na linguagem de especificação, facilitam o processo de manutenção da especificação, pois ajudam a torná-la entendível.

Validabilidade: Especificações escritas obedecendo as regras semânticas pré-definidas na linguagem de especificação, são possíveis de serem avaliadas com relação a seu conteúdo pois ajudam a torná-la entendível.

Critério: Completeza com relaçao ao roteiro definido pela organizaçao

Objetivo: Confiabilidade conceitual
Fator: Suficiência
Sub-fator: Completeza

Definiçao:

A especificaçao possui o critério **completeza com relaçao ao roteiro definido pela organizaçao** na medida em que os aspectos a serem especificados sao baseados num roteiro pré-definido.

Discussao:

A especificaçao deve apresentar a usuários e desenvolvedores uma descriçao completa e objetiva das condiçoes a que deve obedecer o software que vai ser construído. Para que isto aconteça, deve ser estabelecido um roteiro para elaboraçao da especificaçao. As especificaçoes produzidas deverao obedecer a este roteiro.

Existe a possibilidade de que uma determinda seçao do roteiro nao seja aplicável ao problema. Nesse caso, deve ser incluída uma breve explicaçao do motivo pelo qual nao é aplicável.

Processo de avaliaçao:

Este critério pode ser avaliado através das respostas ao seguinte questionário:

	SIM/NAO 1 ou 0
1. todas as funçoes a serem desempenhadas pelo software, estao definidas?	___
2. todas as necessidades de informaçao estao identificadas?	___
3. todas as interfaces com o usuário estao identificadas?	___
4. todas as interfaces com o ambiente de hardware estao identificadas?	___
5. todas as interfaces com outros softwares estao identificadas?	___
6. todas as características relativas ao desempenho do software estao identificadas?	___
7. todas as categorias possíveis de tratamento de erros e exceçoes, por falhas de hardware e software, estao identificadas?	___
8. todas as características de qualidade exigidas pelo usuário estao identificadas?	___
9. todas as características de qualidade inerentes ao software sao identificadas?	___
10. outros requisitos necessários estao identificados?	___
11. todas as referências estao definidas de forma precisa na especificaçao?	___

Critério: Completeza com relação ao roteiro definido pela organização

Objetivo: Confiabilidade conceitual
Fator: Suficiência
Sub-fator: Completeza

Interpretação dos resultados:

Rever a especificação caso existam perguntas com respostas NÃO.

Atributos de qualidade relacionados:

Viabilidade de Implementação: Só é possível implementar adequadamente o produto que se especifica, se o conteúdo da especificação atende às necessidades dos usuários, quando está completa.

Anexo 3

Critério: Necessidade dos requisitos Objetivo: Confiabilidade conceitual
Fator: Suficiência
Sub-fator: Necessidade

Definição:

A especificação possui o critério **necessidade dos requisitos** na medida em que na especificação sejam tratados os requisitos considerados obrigatórios e/ou desejáveis.

Discussão:

Uma especificação de requisitos deve conter apenas os aspectos considerados imprescindíveis, evitando assim descrições desnecessárias e irrelevantes.

Para verificar o grau de necessidade dos requisitos especificados, é conveniente classificá-los em três grupos:

- **requisitos imprescindíveis:** são aqueles requisitos considerados como obrigatórios. Caso não constem na especificação, implicaria na não aceitação do software;
- **requisitos desejáveis:** são aqueles requisitos que ajudam a obter um software mais abrangente. Caso não constem na especifcação não implicaria na não aceitação do software;
- **requisitos opcionais:** são aqueles requisitos que poderão ou não ser considerados.

Processo de avaliação:

Este critério pode ser avaliado através da seguinte escala:

1	os aspectos especificados são considerados obrigatórios.
1	os aspectos especificados são considerados desejáveis.
0.5	os aspectos especificados são considerados opcionais.
0	os aspectos especificados são considerados inúteis.

Interpretação dos resultados:

Se o resultado da avaliação for menor que 0.5, a especificação deve ser revista. Caso existam requisitos opcionais, eles deverão ser considerados depois dos obrigatórios e/ou desejáveis.

Atributos de qualidade relacionados:

Concisão: A inexistência de aspectos irrelevantes na especificação minimiza o volúme de texto da especificação.

Anexo 4

Objetivo : Utilizabilidade
Fator : Reutilizabilidade
Sub-fator: Aplicabilidade

Definição: É a característica de uma especificação de requisitos poder ser reutilizada ao se especificar outra aplicação na mesma área do domínio do problema.

Avaliação

Este sub-fator é avaliado através dos critérios:

- Padronização de termos técnicos;
- Coesão;
- Acoplamento;
- Organização Global.

PARADIGMAS PARA O PROCESSO DE PRODUÇÃO DE SOFTWARE E SUAS CONSEQUÊNCIAS SOBRE OS AMBIENTES DE DESENVOLVIMENTO

SUELI MENDES; ANA REGINA ROCHA
Universidade Federal do Rio de Janeiro - COPPE
C.P. 68511; CEP 21945 - Rio de Janeiro - Brasil

RESUMO

Este trabalho discute paradigmas para o processo de desenvolvimento de software. Tendo em conta estes paradigmas, são identificadas e discutidas, características para ambientes de desenvolvimento de software.

1. INTRODUÇÃO

Antes de discutirmos o que consideramos os paradigmas para ambientes de desenvolvimento de software é conveniente deter-nos um pouco no significado da existência de paradigmas em ciência.

A discussão em torno de paradigmas tem-se dado de diferentes formas ao longo da história da ciência. Pode-se, contudo, observar uma tendência no sentido de que a ciência mais em evidência em um determinado momento trace paradigmas para as demais ciências. Como exemplo podemos observar o que aconteceu ao se considerar a matemática como um paradigma para todas as disciplinas que desejassem ser consideradas como ciência: outras áreas do conhecimento, para serem consideradas ciência, passaram a incorporar métodos matemáticos às suas metodologias de trabalho.

Para exemplificar consideremos o trabalho de Kant

"Prolegômenos a toda Metafísica futura que queira se constituir como ciência" e o trabalho de Karl Popper [7]. Ambos colocam a Física como paradigma. Popper, com seu princípio de refutação, pretende demonstrar que nem o Marxismo nem a Psicanálise de Freud podem ser consideradas ciências.

Ao considerarmos o processo de desenvolvimento de software podemos dizer que este se realiza de acordo com um dos três paradigmas que descrevemos a seguir:

1. **Abordagem artesanal**, onde a construção de software é feita por processos artesanais, sem utilização de métodos. Desta forma, o produto gerado é visto como uma obra pessoal, quase como uma obra de arte. A regulamentação jurídica, atual, usando a lei de direito autoral é uma manifestação da aplicação deste paradigma. Um ambiente de desenvolvimento de software usado neste contexto, não chega a ser algo determinante para o produto mas, simplesmente, um instrumento que o "artista" usa quando lhe é conveniente. Neste trabalho não consideramos este paradigma.

2. **Abordagem formal**, onde se procura aplicar métodos matemáticos, mais especificamente da Lógica, ao processo de desenvolvimento de software, considerando o que é bom para matemática como adequado e útil para o desenvolvimento de software.

Este paradigma conduz a um interesse de pesquisa por tudo aquilo que represente métodos formais de desenvolvimento (linguagens formais para especificação, verificação formal de

programas,etc). Ambientes de desenvolvimento construídos segundo este paradigma deverão ter ferramentas que auxiliem na utilização destes métodos.

Parodiando Kant, poderíamos escrever os "Prolegômenos para todo o software futuro que queira se constituir como ciência", ficando em aberto a questão de se as conclusões seriam tão negativas para o software como o foram para a metafísica. Que a pretensão existe não há dúvida. Neste contexto podemos citar a introdução de H.P.Ershov no "International Symposium on Theoretical Programming", onde declara:

> "Theory of computing, mathematical foundation of computer science, theory of programs, theoretical programming - are not a complete list of identifiers used. However, it is more and more recognized that these identifiers are to a certain degree synonyms signifying that a new mathematical discipline has come into beeing" (Lecture Notes in Computer Science,5, Springer Verlag,1974).

3. Abordagem empírica, onde o processo de desenvolvimento de software é visto de forma similar ao processo de desenvolvimento de produtos de Engenharia. Seguindo este paradigma, a pesquisa será na direção de buscarem-se métodos e técnicas que aproximem, o mais possível, o processo de desenvolvimento de software das características de desenvolvimento de produtos em áreas tradicionais da Engenharia.

Uma das preocupações decorrentes da aplicação deste paradigma é, por exemplo, a busca de se conseguir uma maior visibilidade do produto desde as suas fases iniciais de desenvolvimento. A pesquisa em torno de protótipos rápidos se situa neste contexto. A referência a sistemas tolerantes a falhas é outro exemplo de aplicação deste paradigma.

Ambientes de desenvolvimento de software, construídos neste contexto, procuram fornecer métodos e ferramentas análogos aos das disciplinas de Engenharia.

O processo de desenvolvimento de software é, contudo, uma atividade ainda bastante nova e pouco madura. Neste sentido é, perfeitamente, natural que se busquem paradigmas que orientem este processo. Tem-se, então, a tendência a buscar estes paradigmas em outras áreas do conhecimento. O desejável, entretanto, é que sejam encontrados modelos próprios que reflitam as características originais deste processo. Estes modelos, seguramente, conterão elementos presentes nos três paradigmas descritos acima.

Na medida em que desenvolver software significa encontrar solução para problemas, que terminarão sendo resolvidos através de algoritmos, pode-se afirmar que desenvolver software está muito próximo do processo de resolução de problemas em matemática. Outras atividades, no entanto, (por exemplo, questões relacionadas a confiabilidade e eficiência das soluções encontradas) são próximas a atividades de áreas tradicionais da engenharia. Isto nos leva a afirmar que um ambiente ideal deve considerar todos estes aspectos.

2. CARACTERÍSTICAS PARA AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

Vários ambientes de desenvolvimento de software vem sendo propostos nos últimos anos. Muitos destes, não chegaram a ser realmente úteis. Consideramos que antes de se pretender construir um novo ambiente é importante definir que características são desejáveis para ambientes de desenvolvimento de software e como estas características se relacionam com os paradigmas identificados.

2.1 O ambiente deve fornecer apoio durante todo o processo de desenvolvimento do software.

Embora não haja um consenso sobre como dividir o ciclo de vida em fases (e, nem mesmo, se é desejável definir fases) [3], [4], esse consenso é facilmente obtido no que se refere à necessidade de apoio durante todo o desenvolvimento, independentemente de como ele se faça.

A experiência tem mostrado que é de pouca utilidade utilizarem-se métodos e ferramentas de auxílio ao desenvolvimento de software que dêm apoio a apenas parte do ciclo de vida (continuaremos a usar este termo apesar da polêmica em torno). Estas metodologias e ferramentas tendem a ser de difícil integração com outras, o que traz uma série de problemas ao longo do desenvolvimento.

Um ambiente de apoio ao desenvolvimento de software deve, portanto, auxiliar durante todo o processo de desenvolvimento.

A consideração sobre o desenvolvimento de software em

fases de um ciclo está mais diretamente relacionada ao enfoque empírico que descrevemos acima. No entanto, um ambiente que apoie todo o ciclo de vida pode conter ferramentas que o aproximem do enfoque formal, por exemplo um verificador formal de programas. No entanto, se acreditamos na possibilidade de provar formalmente a correção de um software, por mais complexo que ele seja, fases do ciclo de vida tradicional tais como testes e manutenção poderiam ser eliminadas. Assim sendo, o paradigma do enfoque formal modificaria de modo drástico o ciclo de vida do software.

2.2 O ambiente deve conter um ciclo de vida, métodos e ferramentas que apoiem o desenvolvimento

A dificuldade de construir um ambiente de desenvolvimento de software realmente útil e efetivo, está diretamente relacionada à dificuldade de entender o processo de desenvolvimento de software. Entender como se dá o desenvolvimento é essencial ao se propor a construção de um ambiente.

Um ambiente de apoio ao desenvolvimento de software, pode ser definido como composto de:

- um ciclo de vida, definindo as fases do processo de desenvolvimento e as atividades a serem realizadas em cada fase;
- métodos, usados para organizar o pensamento e o trabalho do usuário ao longo do processo de desenvolvimento, e,
- ferramentas, automatizadas ou não, que tornam possível utilizar os métodos.

Diferentes aplicações têm características diferentes. Assim sendo, seus processos de desenvolvimento também têm diferentes características. Neste sentido, características de aplicações devem determinar a escolha de um ou outro método. Um ambiente deve conter diferentes tipos de ferramentas: ferramentas para descrição do problema (linguagens de representação e editores), ferramentas para avaliação da qualidade e ferramentas para apoio à gerência (estimadores de custo, acompanhadores de projetos, etc).

2.3 O ambiente deve conter várias linguagens formando uma hierarquia, cada uma delas apropriada para descrever o produto em um determinado nível de detalhe

Durante o processo de desenvolvimento, são geradas diversas representações do software. Em cada uma delas, o produto é descrito com um determinado objetivo. Assim, tem-se representações cujo objetivo é possibilitar a interação com o usuário para fins de definição do produto a ser desenvolvido. Em outros momentos, as representações geradas têm outros objetivos (por exemplo a comunicação com o projetista ou com o implementador) até se chegar à representação final (o software-produto) cujo objetivo é atender aos requisitos do usuário que motivaram sua construção.

Estas representações são produzidas utilizando-se linguagens e a qualidade do produto final dependerá da qualidade e adequação destas linguagens. Existem, atualmente, várias linguagens propostas. Algumas abordagens pretendem ter linguagens que sejam adequadas para uso durante todo o

processo de desenvolvimento, como por exemplo as linguagens ADA [5] e GYPSY [1]. Outras abordagens propõem uma sequência de linguagens a serem usadas ao longo do desenvolvimento, como por exemplo o ambiente HDM [9].

Condideramos mais adequado ter um ambiente que contenha um conjunto de linguagens, cada uma apropriada para descrever o produto em um determinado nível de detalhe. Estas linguagens deverão formar uma hierarquia, onde as linguagens de nível mais alto serão usadas no início do desenvolvimento e terão como objetivo favorecer a interação com o usuário, seja por seu alto grau de comunicabilidade ou por possibilitarem a construção de protótipos. No primeiro caso, as pesquisas em torno do processamento de linguagens naturais, em Inteligência Artificial, poderiam levar ao nível ideal de comunicação (guardadas as devidas restrições). Veja-se, como exemplo, SAFE [2].

O principal problema do uso de diferentes linguagens está na sua integração. A grande motivação de se usar uma única linguagem, como no projeto ADA, reside certamente nesta questão. No entanto, se tivermos ferramentas para auxiliar nesta integração, o problema estaria em parte resolvido, o que nos remete ao item abaixo.

2.4 O ambiente deve fornecer auxílio semi-automatizado, para migrar de uma linguagem na hierarquia para outra.

Migrar de uma linguagem para outra na hierarquia não é uma tarefa trivial. O ambiente deve conter métodos que indiquem,

claramente, o processo e os passos a serem seguidos. Idealmente deve, pelo menos, fornecer um apoio semi-automatizado para a realização desta tarefa.

Este apoio deve possibilitar caminhar na hierarquia nas duas direções. Embora se tenha claro que desenvolvimento "top-down" é a melhor estratégia para se controlar a complexidade, sabe-se que o desenvolvimento de software dificilmente se dá de uma forma puramente "top-down". De forma similar à Engenharia é comum, ao longo do processo de desenvolvimento de software, a necessidade de modificações e estas exigem o uso da estatégia "bottom-up".

2.5 O ambiente deve conter linguagens que permitam descrever o mesmo problema sob diferentes pontos de vista permitindo, assim, detectar inconsistências e conflitos.

O ambiente pode, ainda, conter mais de uma linguagem para um mesmo nível na hierarquia. Estas linguagens possibilitariam descrever o produto sob mais de um ponto de vista (por exemplo, atividades e dados [8]), tornando possível a detecção de inconsistências e conflitos.

Esta característica é especialmente útil se existirem mecanismos automatizados de verificação da consistência entre as descrições.

2.6 O ambiente deve conter um método para avaliação da qualidade do produto (validação) e do processo de desenvolvimento (verificação).

Um ambiente para desenvolvimento de software deve conter

ferramentas para controle da qualidade tanto no que se refere aos produtos gerados (validação) quanto ao processo de desenvolvimento destes produtos (verificação). Estas ferramentas, por sua vez, devem estar baseadas em algum método para avaliação da qualidade que nos permita confiar nas avaliações realizadas.

2.7 O ambiente deve ser de uso amigável, guiando o usuário através de um determinado método e auxiliando-o no uso das ferramentas.

Sabe-se, na prática, que forçar os usuários a usar métodos novos e desconhecidos para eles é tarefa árdua e, em alguns casos, impossível. Um exemplo disso, é tentar forçar o analista e o programador sem maturidade matemática a usar métodos formais. No entanto, se desejamos melhorar a qualidade do software essa tarefa deve ser empreendida de alguma forma.

Acreditamos que um modo de atacar o problema é oferecer ambientes com métodos e ferramentas de mais fácil uso, que convençam o usuário mais recalcitrante da vantagem de usá-los e procurar, então, fazê-lo tentar o uso de instrumentos mais sofisticados. O problema de qualquer forma é de difícil solução. E a única maneira que divisamos de uma possível solução, é investir na educação da próxima geração formando engenheiros de software com novos hábitos.

2.8 O ambiente deve conter uma biblioteca de componentes de forma a permitir a re-utilização de especificações, projeto e

código.

A re-utilização de módulos é algo que traz tantas vantagens que, em geral, não está sujeita a grandes polêmicas. Pode-se, é claro, discutir se se deve levar isto como uma meta. Módulos podem ser construídos com o objetivo de serem re-utilizados em diferentes tipos de aplicações. Estes módulos tendem a ser demasiado gerais o que pode prejudicar a eficiência. Se este é um requisito importante para o produto, a questão da re-utilização pode ter discutida a sua aplicabilidade.

Outro aspecto a ser considerado é a conveniência de se re-utilizar, não apenas módulos, mas também especificações e projeto. Veja-se, neste caso, o sistema DRACO [6]. Para isso são necessárias bases de conhecimento sobre o domínio das aplicações o que nos remete a outra característica (incorporação de técnicas de Inteligência Artificial ao processo de desenvolvimento de software).

2.9 O ambiente deve permitir a realização de verificações formais e/ou a construção de protótipos rápidos.

Este item está de acordo com a nossa posição de que um ambiente de desenvolvimento de software deve ter seu modelo próprio sem se conformar exclusivamente a nenhum dos paradigmas citados. Isto, entretanto, não quer dizer ficar "em cima do muro" com relação às posições radicais. Acreditamos, realmente que existe espaço e, principalmente, diversidade de aplicações que podem acomodar instrumentos incorporados dos vários paradigmas.

2.10 O ambiente deve incorporar técnicas de Inteligência Artificial ao processo de desenvolvimento de software.

Consideramos, aqui, a possibilidade do casamento entre Engenharia de Software e Inteligência Artificial e não apenas da Engenharia de Software com Sistemas Especialistas [10].

A união entre Inteligência Artificial e Engenharia de Software pode ser bastante frutífera. De um lado Inteligência Artificial já está motivando novas linhas de pesquisa em Engenharia de Software, tanto do ponto de vista do próprio ciclo de vida quanto das metodologias a serem desenvolvidas e aplicadas, com vantagens para ambos os lados. Por outro lado, ferramentas de Inteligência Artificial vem sendo e deverão continuar sendo utilizadas em Engenharia de Software, o que nos parece ter um grande potencial para se conseguir sair do longo túnel, apelidado "crise de software".

Quanto ao problema da utilização de Sistemas Especialistas em Engenharia de Software, acreditamos que sua validade é, ainda, como pesquisa acadêmica e que é prematuro se partir, neste momento, para a construção de sistemas especialistas com possibilidades de lançamento no mercado.

2.11 O ambiente deve ser implementado de forma incremental, de modo que as primeiras ferramentas desenvolvidas possam ser usadas e criticadas por seus usuários possibilitando melhorar a qualidade do ambiente.

Desenvolvedores de ambientes costumam recomendar que estes sejam desenvolvidos de forma incremental. Concordamos com esta abordagem pois acreditamos ser melhor partir de algo

simples e concreto, mas que possa ser utilizado em prazo relativamente curto do que partir, logo de início, para grandes metas exigindo longos prazos.

3. CONCLUSÕES

Procuramos discutir, aqui, o efeito que a adoção de um determinado paradigma para a atividade de desenvolvimento de software pode ter na criação de ambientes.

Tentamos detectar posições extremadas que podem levar à exclusão de métodos e ferramentas em detrimento de outras ao se adotar um determinado paradigma.

Existem várias razões fortes para considerar que qualquer posição radical, neste sentido, pode ser prejudicial:

- métodos formais tem sido grandemente pesquisados, mas ainda não foram demonstrados, para software grandes e complexos, como sendo realmente exequíveis. A esperança reside no desenvolvimento de métodos automáticos ou semi-automáticos, possivelmente, derivados de técnicas de Inteligência Artificial (como provadores automáticos de teoremas);
- protótipos representam pesquisas recentes e a demonstração cabal de que o esforço dispendido para realizá-los é compensado em termos de custos do desenvolvimento, não foi ainda conseguido. Por outro lado, as transformações que os protótipos devem passar nos vários níveis desde a primeira especificação até o sistema totalmente codificado ainda não estão claramente definidos. Muito esforço de pesquisa e experimentação

está por ser feito;

- tanto a utilização de métodos formais como de construção de protótipos demandam esforços dos usuários que exigem persuação, demonstração de sua utilidade, etc. Assim, pode-se considerar que para projetistas de sistemas de tempo-real, por exemplo, a demonstração da utilidade da construção de protótipos é muito mais fácil do que convencê-lo a utilizar métodos formais baseados em Lógica Temporal.

Em resumo, o que propomos, aqui, é que se caminhe no sentido de encontrar um novo paradigma como modelo para a atividade de desenvolvimento de software. Este novo paradigma deve incorporar elementos de vários paradigmas encontrados ao longo da história da Ciência e da Tecnologia, uma vez que não acreditamos na total ruptura com o passado. E aí adotamos o paradigma do evolucionismo de Darwin, que tanto influenciou várias ciências, mas no nosso caso temperando-o com a possibilidade de saltos de vez em quando.

REFERÊNCIAS

[1] A.L.Ambler et al "GYPSY: a language for specification and implementation of verifiable programs"; In Proceedings of the ACM Conference on Language Design for Reliable Software; março 1977

[2] R.Balzer et al "Informality in program specifications"; IEEE Transactions on Software Engineering, vol-SE-4, n 3, Julho 1978

[3] B.I.Blum "The Life Cycle: a debate over alternate models", ACM SIGSOFT Software Engineering Notes, vol 7, n 4, outubro 1982

[4] G.R.Gladden "Stop the life cycle, I want to get off, ACM SIGSOFT Software Engineering Notes, vol 7, n 2, abril 1982

[5] Ichbiah et al " Ada Programming Course, ALSYS, La Cella Saint Cloud, França, dezembro 1980

[6] J.M.Neighbors Software Development using Components: Tese de Doutorado, Universidade da California, Irvine, 1981

[7] K.Popper The Logic of Scientific Discovery; Routledge and Kegan Paul, Londres, 1959

[8] D.T.Ross "Structured Analysis (SA): a language for communicating ideas"; IEEE Transactions on Software Engineering, vol SE-3, n 1, Janeiro 1977

[9] B.A.Silverberg "An overview of the SRI hierarchical development metodology"; in Software Engineering Environments; Hunke,H. ed.; North Holland; 1981

[10] W.T.Znaikerman "Expert Systems and software Engineering: ready for marriage?"; IEEE EXPERT; Inverno 1986

Dos aplicaciones de las tablas de decisión

Autor: Luis Roberto Ojeda Ch.
Ingeniero, M.S. en Computación
U. C. L.A.
Universidad Nacional de Colombia
Departamento de Sistemas
Bogotá

Las tablas de decisión como instrumento de programación han sido objeto de estudio para los investigadores en forma un tanto irregular, apareciendo nuevos desarrollos concentrados en determinados períodos. Hacia la mitad de los años setenta es frecuente encontrar uno o dos artículos sobre el tema en cada entrega de las "communications of the ACM".
En los dos últimos años sólo aparece un artículo sobre tablas de decisión en la misma publicación. Para los amantes del tema esto no puede representar obsolescencia definitiva de la técnica. Muchas horas hay invertidas en ellas muchas formas ingeniosas se han producido.
El presente artículo presenta una introducción de las tablas en la programación de autómatas. Hay la esperanza mínima de que esta forma complete el panorama corriente de representación de autómatas. Se desea en el futuro hacer la evaluación de las opciones de codificación de los autómatas.
La segunda parte del artículo trata la conversión de programas a tablas de decisión. Esto se debe ser mejor que el tratamiento tradicional de convertir el programa en un diagrama de flujo, por razones de manejo de espacio en el medio físico de salida, y de tiempo de ejecución.
Es importante mencionar que esta aplicación sigue los lineamientos de una publicación sobre el problema (John Cavoaras, University of Glasgow) y que las proposiciones condicionales se reducen a la forma lógica , para mayor claridad. El desarrollo formulado aquí pretende dar un punto de partida consistente para el análisis de las posibilidades avizoradas en el artículo de cavoaras, reseñado dentro de este trabajo.

Primera Aplicación

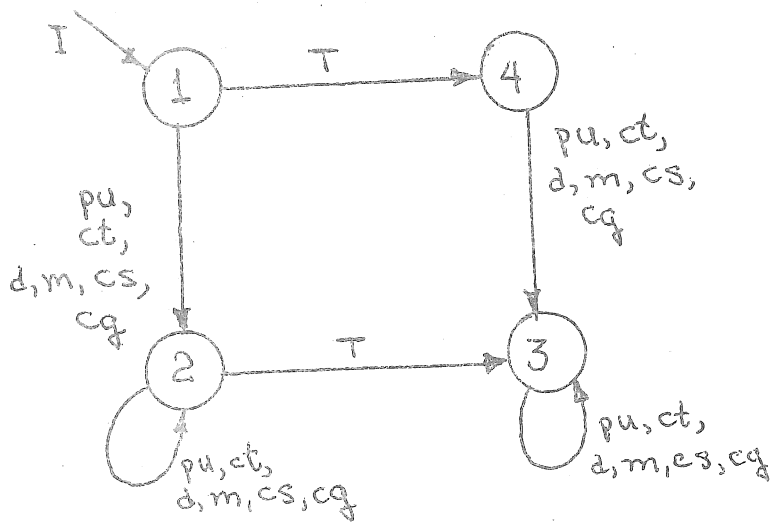
Conversión de un autómata determinístico finito en una tabla de decisión simple.
Aparte de su uso como analizador léxico, el autómata puede realizar funciones de reconocimiento en otros ambientes.
El siguiente caso servirá para ilustrar el procedimiento de conversión.

El artículo 59 de la Constitución de la República de Colombia establece:
"Para ser elegido contralor general de la república se requiere ser colombiano de nacimiento y en ejercicio de la ciudadanía; tener más de 35 años de edad; tener título universitario en derecho o en ciencias económicas o financieras. Además, haber desempeñado en propiedad alguno de los cargos de Ministro del Despacho, Magistrado de la Corte Suprema de Justicia, Consejero de Estado, Contralor General de la República; o habe sido Miembro del Congreso Nacional por lo menos durante cuatro años, o Profesor Universitario en las cátedras de ciencias jurídico económicas, durante un tiempo no menor de cinco años".

Para obtener un autómata simplificado, asumiremos el siguiente alfabeto:

T	:	Implica el cumplimiento de: Más de 35 años de edad. Colombiano de nacimiento Ejercicio de la ciudadanía Título en derecho o en ciencas económicas o financieras
d	:	Ministro del despacho (en propiedad)
m	:	Magistrado (en propiedad)
cs	:	Consejero de estado (en propiedad)
cg	:	Contralor general (en propiedad)
ct	:	Miembro del Congreso Nacional ($\geq$ 4 años)
pu	:	Profesor universitario jurídico-económicas ($\geq$ 5 años)

El autómata será:



El procedimiento propuesto para la conversión es el siguiente:

I) Tomar el estado de partida (estado I) del autómata.

II) Formar una condición múltiple con signos del alfabeto.
(Condición múltiple: Es un conjunto de condiciones sencillas.
Cada condición sencilla es un subconjunto del alfabeto que marque algún(os) arco(s) del autómata).
Si de un estado hay salida a más de un estado formar la condición K = Estado.
Formar siempre (de todas maneras) la condición K = Estado para el estado de aceptación (Estado Final).
Si hay un número n de estados cada uno con salida solamente a un estado, dar lugar para la condición K = Estado para n - 1 de tales estados.
El número de condiciones es igual al número de marcas diferentes en los arcos, más el número de estados con más de un estado de transición. Si hay estados (n) con transición a un sólo estado, se suman n-1 condiciones.

III) Acciones
Formar una acción K = Estado con cada estado.
(una acción por cada estado del autómata)"

IV) Acciones
Formar las acciones: Volver (para regresar a aplicar la tabla); Para 1 (para aceptar) y Para 2 (para recharzar).

V) Reglas de decisión
Formar - por cada estado con más de una salida - una regla de decisión para cada arco que marque una transición desde ese estado.
Si de todo estado hay más de una salida el número de reglas de decisión es igual al número de arcos.
Para (n-1) de los n estados con sólo una salida aplicar el mismo concepto anterior -V-

LLenado de la tabla

I) U = I (estado de partida); nn = 0
k = primer arco saliente de I

II) (Condiciones).
Marcar S en la casilla K = valor de U
Marcar S en la casilla identificada por k.
Marcar N en el resto de casillas de condiciones
(Acciones).
Marcar (X) en la casilla correspondiente (K =) al estado al que se hace transición con éste arco.
Marcar (X) la casilla correspondiente a 'volver'
Aplicar con todos arcos salientes del estado actual el paso 2 para formar diferentes reglas de decisión.
variark
Tomar un nuevo estado conectado directamente con los ya procesados.
Si de él sale mas de un arco, hacer U = Estado k = primer arco saliente. Aplicar el paso 2.

Si del estado sale solamente un arco y nn $\neq 0$
formar en la s condiciones la condición K = Estado; Hacer U = estado, k = arco saliente.
Aplicar el paso 2.
Si el estado sale solamente un arco y nn = 0, encontrar el estado al que se transita. Si el tránsito es al mismo estado, buscar otro nuevo estado conectado directamente a los ya procesados. Si no hay, seguir como si el tránsito fuese a otro estado .
Si el tránsito es a otro estado, suponer que éste arco sale y regresa en este nuevo estado. Verificar que no se repite un arco. Si es así, eliminar la repetición k = primer arco saliente.
Formar las reglas correspondientes a este estado según el paso 2, excepto que no se marcará S en la casilla (K = valor de U) y quedará en blanco.
nn = 9

Para cerrar

Al finalizar con los estados, formar la regla de aceptación negando todas las condiciones sencillas (condición múltiple) y afirmando la condición (K =) (estado final). En las acciones marcar (X) Parar 1.
Formar la regla de rechazo asumiendo cualquier conjunto de condiciones no contempladas en las reglas anteriores, y marcar (X) la acción Parar 2.

La aplicación del procedimiento origina la siguiente tabla de decisión para el autómata definido antes:

pu, ct, d, m, cs, cg	N	S	S	N	S	N	O
T	S	N	N	S	N	N	T
K = 1	S	S	N	N	N	N	R
K = 2	N	N	S	S	N	N	O
K = 3	N	N	N	N	-	S	S
K = 1							
K = 2		X	X				
K = 3				X	X		
K = 4	X						
Volver	X	X	X	X	X		
Parar 1						X	
Parar 2							X

Las formas conocidas de representación para un autómata determinístico finito incluyen:

a) Una estructura de listas
Gries, Compiler Construction for digital computers, J. Wiley, 1971 Ch. 3
Cada estado con K arcos salientes se representa con 2K+2 palabras. La primera contiene el nombre del estado. La segunda el valor de K.
Cada par subsiguiente contiene un marcador de arco y un apuntador al comienzo de la representación del estado al que se transfiere.
No hay indicación de eficiencia.

b) Una cláusula CASE (cuando sea disponible en el lenguaje de representación del autómata)
Waite y Goos (Compiler Construction, Springer Verlag 1984 Ch. 6) establecen solamente que el costo es considerable con esta alternativa. Lo comparan con la matriz de transición (generalmente costoso en términos de memoria) y sólo reconocen estas dos posibilidades para representar el autómata.

c) La matriz de transición

```
While (símbolo todavía incompleto) do
estado: = tabla [ estado, nuevo caracter]
```

Aho y ullman (principles of Compiler Design, Addison Wesley 1979 Ch. 3) señalan que éste arreglo bidimensional indizado con estados y caracteres es el más rápido pero puede consumir mucho espacio [varios centenares de estados - filas - por 128 caracteres - columnas -]
Los autores citados en b) y c) presentan formas de comparación de los métodos conocidos.
La facilidad Lex de UNIX proporciona el tratamiento de matriz de transición.
Para ubicar el tratamiento propuesto en este artículo se resalta : a) su parentesco con la matriz de transición, basada en estados y caracteres como entradas para recorrer los estados.
b) La variedad de formas de programar una tabla de decisión para sacar ventajas de su estructura o de los datos que comunmente maneja.
Para tener sólo una posibilidad interesante ver: "Extending the information theory Approach to Converting Limited- entry Decision Tables to Computer programs".
Keith Shwayder, Comm ACM Sept. 1974 Vol. 17 nr 9 p. 532.
(El objetivo es minimizar el tiempo de ejecución del programa resultante).

c) Finalmente, puesto que la conversión de una tabla de decisión en un programa es algorítmica, puede pensarse en variar las posibilidades para un mismo autómata.

Segunda aplicación

Conversión de programas a tablas de decisión
Lograr un procedimiento eficiente para convertir un programa (escrito en un lenguaje particular) en una tabla de decisión es un objetivo seductor: en primer término, la tabla de decisión, que

es un vehículo natural de comunicación, se convierte en un sustituto ventajoso del diagrama de flujo (todavía útil en programación) por cuanto el tamaño del programa para lograr una tabla de decisión a partir de un código es necesariamente inferior al que corresponde a la obtención de un diagrama de flujo.
En segundo lugar, existiendo métodos diversos para convertir automáticamente una tabla de decisión en un programa de computador, y algoritmos para optimizar una tabla (information theory applied to the conversion of decision tables to computer programs S. Ganapathy, V. Rajaraman, Comm. ACM Sept 1973) (The development of Decision Tables via parsing of Complex Decision situations, Comm ACM, Jun 1973) se puede pensar en la conversión con el propósito de optimizar y eventualmente, por lo menos como ejercicio teórico, con el objetivo de disponer de codificación en otro lenguaje.
El procedimiento que viene enseguida está fundamentado en el trabajo de John Cavouras (On the conversion of programs to Decision Tables: Method and Objectives Comm. ACM Agosto, 1974) en este trabajo los objetivos propuestos son depuración del programa y optimización. Aunque Cavouras estima alcanzado el objetivo pese a que en algún caso un programa puede conducir a dos tablas diferentes, lo que le lleva a concluir que las tablas de decisión no son muy útiles para determinar la semántica de programas de computación, vale la pena un replanteamiento de base del procedimiento de conversión, con el propósito de analizar mejor el planteamiento teórico y las referencias bibliográficas del trabajo citado.
El procedimiento fué desarrollado pensando en que las proposiciones condicionales usadas en el programa son solamente del tipo IF lógico, para simplificar.

Procedimiento

1) K = 1 identificador de regla de decisión
Símbolo = S
ra = 1 rótulo actual de condición
rac = 1 rótulo actual de acción

2) Visitar y marcar las proposiciones del programa hasta encontrar una proposición condicional .
Marcarla.

3) Proposición condicional
Llenar el rótulo actual de condición (ra)
Abrir una columna (regla de decisión K) y frente al rótulo actual de condición llenar con S.
Poner en la pila la dirección de la rama N y el valor de ra (para recordar la condición que se niega por éste camino).
Seguir la trayectoria S.

4) Si encontramos una proposición condicional, marcarla ra = ra+1
Llenar el rótulo actual de condición
Llenar la entrada correspondiente (columna K) con S
Poner en la pila la dirección de la rama N y el valor de ra.

Si encontramos otra proposición condicional, repetimos 4.
Si encontramos una acción, llenamos el rótulo actual de acción y la entrada correspondiente en la columna K con X, marcamos la acción, hacemos rac = rac+1, y repetimos con las acciones que prosigan hasta encontrar un Alto.

5) Cuando aparece un Alto sacamos de la pila: ra(p) (tomado de la pila) identificará la línea de condición.
Si ra(p) = 1 hacer símbolo = N
Hacemos K = K + 1 Abrimos una nueva columna (columna K)
Sea T: tabla de condiciones (parte superior de la tabla de decisión).

En T(ra(p), K) escribimos N
Si ra(p) > 1 Hacemos:

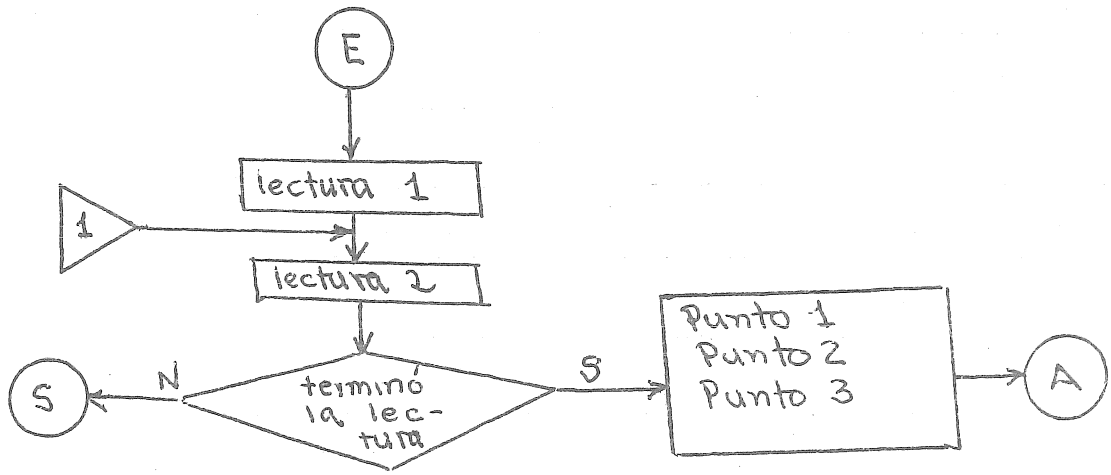
n = 1
Si T(ra(p) - n , K - 1) ≠ b
Hacemos T(ra(p) - n , K) = T(ra(p) - n, K - 1)
de otro modo hacer n = n+1 y repetirnos este paso.

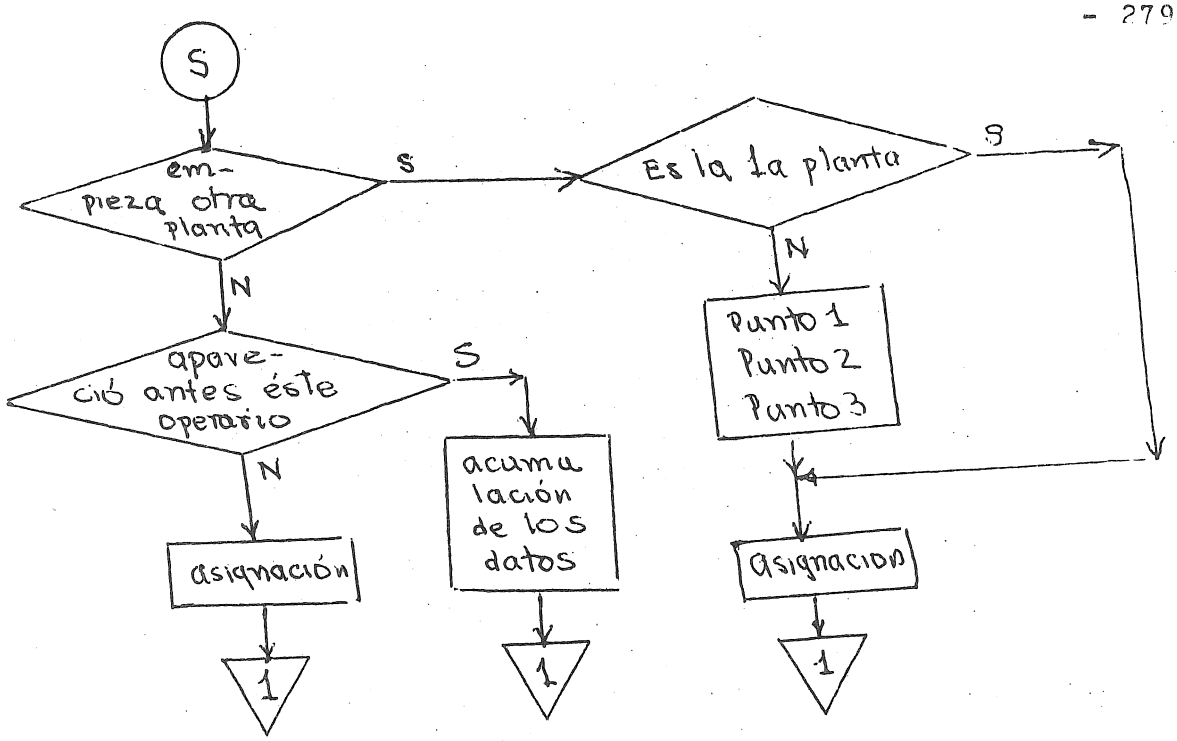
Si ra(p) > 2

Hacemos T(nn, K) = Símbolo
para nn desde 1 hasta ra(p) - 2

Con la dirección de la instrucción (rama N) pasamos a 4.

Un Alto es una proposición condicional que aparece luego de una o mas acciones.
Si esta proposición condicional ya está marcada y es la primera de la tabla, se adiciona la acción volver antes de proceder con 5.
De otro modo se adiciona la acción 'ejecutar nueva tabla', antes de proceder con 5.
La nueva tabla comenzará con la proposición condicional que originó el Alto.
Es más fácil seguir el procedimiento utilizando un diagrama de flujo en lugar del código correspondiente en algún lenguaje.





La tabla de decisión resultante es:

Terminó la lectura?	S	N	N	N	N
Empieza otra planta?		S	S	N	N
Es la 1ra planta?		S	N		
Apareció antes éste operario?				S	N
Punto 1	X		X		
Punto 2	X		X		
Punto 3	X		X		
Alto	X				
Asignación		X	X		X
Acc. de datos				X	
Lectura 2		X	X	X	X
Volver		X	X	X	X

Uma Análise Realista da Utilização do Enfoque da Prototipagem Rápida no Desenvolvimento de Sistemas de Informação.

Marcos R. S. Borges

Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Caixa Postal 2324
20001 - Rio de Janeiro
BRASIL

RESUMO: Neste artigo discute-se as condições julgadas necessárias para que as vantagens do enfoque da prototipagem rápida se materializem sem incorrer nos problemas desta abordagem. Estas condições são discutidas no plano das características da aplicação a ser desenvolvida, do tipo de usuário, do ambiente de desenvolvimento e das ferramentas utilizadas no projeto. A posição defendida neste artigo é de que a adoção do enfoque da prototipagem rápida, principalmente quando se refere ao desenvolvimento do sistema através de versões sucessivas, necessita de condições favoráveis a sua utilização, pois sem estas, muitas das vantagens da abordagem são perdidas.

PALAVRAS-CHAVE: Engenharia de Software, Metodologia de Desenvolvimento de Sistemas, Ciclo de Vida, Prototipagem Rápida.

Marcos Roberto da Silva Borges, Engenheiro Eletricista (UFRJ 1975), Mestre em Ciência da Computação (COPPE/UFRJ 1981), Doutor em Ciência da Computação (East Anglia 1986). Analista Consultor do Núcleo de Computação Eletrônica da UFRJ e Professor Adjunto do Departamento de Computação da UFRJ. Trabalha na área de computação desde 1972 e atua junto ao grupo de desenvolvimento de sistemas para a administração da universidade como consultor de apoio nas áreas de Banco de Dados e Engenharia de Software.

1 - INTRODUÇÃO

Muito tem-se discutido sobre as vantagens do enfoque da prototipagem rápida como alternativa ao enfoque tradicional baseado no ciclo de vida em fases [3, 6, 7, 8, 9, 10, 13, 16]. O entusiasmo pelo enfoque de prototipagem rápida é justificável pelas vantagens tão propaladas pelos defensores deste enfoque: retorno mais rápido aos usuários, maior chance de sucesso da aplicação e uma evolução do sistema mais condizente com a necessidade e a capacidade de absorção reais do usuário [9, 10, 11].

A verificação destas vantagens na prática, porém, está condicionada a uma série de características que vão desde o tipo de problema a ser resolvido a formação do pessoal de desenvolvimento, passando pela disponibilidade de ferramentas adequadas. Um exemplo disto é o fato que, de uma forma ou de outra, as idéias em que se baseia o enfoque da prototipagem rápida já são utilizadas há bastante tempo no desenvolvimento de sistemas sem que com isso se tenha alcançado as vantagens tão propaladas. Por força muitas vezes de prazos exíguos ou da formação dos responsáveis pelo desenvolvimento da aplicação, é comum encontrar-se projetos que derrespeitam o ciclo de vida tradicional através da simplificação excessiva das fases de análise e projeto. Nestes casos os fundamentos do enfoque da prototipagem rápida foram adotados empiricamente causando os problemas já conhecidos deste enfoque, qual sejam: documentação deficiente, dificuldades para evolução/manutenção do sistema, etc.

Neste artigo discute-se as condições julgadas necessárias para que as vantagens do enfoque da prototipagem rápida se materializem sem incorrer nos problemas desta abordagem. O artigo se concentra na utilização do enfoque da prototipagem em situações que sugiram o desenvolvimento do sistema através da produção de versões sucessivas. Estas condições são discutidas no plano das caracteristicas da aplicação a ser desenvolvida, do tipo de usuário, do ambiente de desenvolvimento (métodos e formação do pessoal de análise) e das ferramentas utilizadas para o desenvolvimento. Pretende-se com isto contribuir para o esclarecimento da premissa de que a adoção do enfoque da prototipagem rápida necessita de condições favoráveis a sua utilização sem as quais muitas das vantagens desta abordagem são perdidas.

2 - O CICLO DE VIDA TRADICIONAL E ALTERNATIVAS

O ciclo de vida tradicional consiste, em geral, da condução do desenvolvimento do sistema pelo seguinte processo evolutivo [6]:

1 - Análise Organizacional;
2 - Avaliação dos Sistemas;
3 - Análise de Viabilidade;
4 - Plano de Projeto;
(passos 1 - 4 compõem o que se chama de Ante-Projeto)
5 - Projeto Lógico;
6 - Projeto Físico;
7 - Definição de Programas;
8 - Implementação;
9 - Operação;
10 - Revisão e Avaliação.

Dentro deste modelo é que se dá o desenvolvimento da maioria dos projetos de sistemas de informação. As etapas se dão de uma forma sequencial, pois, em geral, o resultado gerado por uma fase é usado como entrada para a seguinte.

Os que defendem do ciclo de vida tradicional consideram que este representa um modelo que auxilia o processo de desenvolvimento do projeto, pois marca bem as etapas que vão sendo cumpridas, e facilita o acompanhamento da evolução deste [8].

Um outro argumento forte em defesa do ciclo de vida tradicional é que a divisão do trabalho nas etapas que o compõe está associada muito fortemente ao nível de detalhe de cada fase [14].

Por outro lado, há vários autores que apontam deficiencias no enfoque tradicional, pois este não induz uma participação maior do usuário no processo de desenvolvimento além de gerar um intervalo muito grande entre a definição do problema e o uso efetivo da solução computacional. Isto provoca um risco do sistema não ser concluido ou, se o for, não estar dentro das reais necessidades dos usuários.

Uma amostragem de dados sobre o desenvolvimento de projetos para o Departamento de Administração do governo americano durante o ano de 1979 mostra que somente cerca de 3% do total gasto com o desenvolvimento de sistemas de aplicação de 6,5 milhões de dólares foram efetivamente aproveitados. Isto demonstra bem os problemas que a má definição de requerimentos provoca.

Várias tentativas foram ou estão sendo feitas no sentido de formalizar melhor o processo de desenvolvimento através do enfoque do ciclo de vida tradicional. Neste

enfoque é que se pode categorizar os trabalhos sobre análise e projeto estruturado de sistemas e os de metodologias para uma ou mais fases do ciclo, como o SADT e o SAMM.

Há cerca de seis anos entretanto, começaram a surgir artigos que vieram em defesa de enfoques alternativos para o processo de desenvolvimento de sistemas de informação. Um destes enfoques é o da Prototipagem Rápida.

3 - PROTOTIPAGEM RÁPIDA: VANTAGENS E PROBLEMAS

Em várias ramos da Engenharia o enfoque de prototipagem é largamente empregado como uma ferramenta poderosa e essencial. Entretanto, no processo de desenvolvimento de software a prototipagem é raramente empregada [18]. Recentemente, um movimento na direção de fazer um maior uso de protótipos foi iniciado e vários artigos e experiencias foram ou estão sendo publicadas.

De um modo geral, o uso de protótipos no processo de desenvolvimento de software se dá em três modalidades:

A primeira é como auxílio na especificação de requerimentos do software. O uso neste caso é recomendado quando o usuário tem pouca experiencia com sistemas computadorizados ou quando se trata de um software totalmente novo do qual, mesmo o usuário mais experiente tem pouco conhecimento. O uso do protótipo viria então ajudar a levantar os requerimentos através da utilização efetiva de uma representação do sistema.

A segunda modalidade visa a utilização do protótipo para efetuar um estudo de viabilidade de parte ou da totalidade do sistema. Neste caso o protótipo é utilizado para avaliar e experimentar soluções para um determinado problema, em geral associado a questão de performance, por exemplo, o tempo de acesso que uma certa organização de dados proporciona.

Finalmente, a terceira modalidade prevê que o protótipo evolua efetivamente para o produto final. Ao contrário das duas situações anteriores, onde o protótipo é descartado após a obtenção dos resultados esperados, nesta terceira modalidade cada versão gerada visa atender as necessidades imediatas do usuário e serve como base as versões seguintes. Desta forma, se dá um processo que é denominado de desenvolvimento através de versões sucessivas.

As vantagens e desvantagens do enfoque variam de acordo com a modalidade em que este é utilizado. Na primeira há a potencial melhoria da especificação do sistema pois os requerimentos são obtidos a partir do uso efetivo de um

modelo que se propõe a executar as funções reais do sistema. Além disso, induz o envolvimento por parte do usuário na fase de definição do projeto, que nem sempre é conseguido pelo processo convencional de entrevistas e discussões sobre modelos hipotéticos. Se alcançados, estes efeitos irão resultar em um sistema de melhor qualidade e de menor custo de manutenção, visto que os requerimentos foram mais consolidados.

Como desvantagem há o custo e o tempo extra dispendido na construção do protótipo. Ao contrário do que se esperaria, o enfoque de prototipagem nesta modalidade aumenta ao invés de reduzir o tempo total de desenvolvimento do sistema.

A segunda modalidade apresenta de uma maneira geral as mesmas vantagens e desvantagens da primeira modalidade. A diferença está na fase do projeto em que o protótipo é utilizado e o que se objetiva com o uso deste.

A terceira modalidade difere das duas anteriores na maioria dos aspectos, pois cada versão do protótipo objetiva o produto final e não somente prover subsídios para um projeto de melhor qualidade. As vantagens do enfoque de versões sucessivas é que este propicia ao usuário uma resposta mais rápida às suas necessidades mais prementes. Como não há necessidade da completeza na elaboração dos requerimentos, é possível através de ferramentas adequadas prover o usuário de uma versão parcial mas real do sistema de forma mais rápida.

A desvantagem é que este processo em geral é bastante dispendioso porque pode ser necessário que uma parte substancial do código seja gerado várias vezes para que este se adapte aos novos requerimentos. Desta forma, embora se possa alcançar resultados parciais em curto espaço de tempo, o produto final pode levar muito mais tempo e ser muito mais caro do que se fosse desenvolvido pelo método tradicional.

Entretanto, há ainda uma outra vantagem que nem sempre é visível para aqueles que não lidam com o problema diretamente. É que muitas vezes, na tentativa de evitar omissões que mais tarde poderão lhe causar problemas, o usuário tende a ser mais abrangente em seus requerimentos do que o necessário. Isto acarreta em muitos casos que dados ou relatórios jamais sejam utilizados pois não fazem parte das necessidades da aplicação. Com o enfoque de versões sucessivas este problema é minimizado pois o projeto evoluiu em função das necessidades reais do usuário, reduzindo desta forma o tamanho do sistema.

Neste artigo, somente o uso do enfoque na terceira modalidade será abordado.

4 - CONDIÇÕES PARA UTILIZAÇÃO DO ENFOQUE

A disciplina Engenharia de Software mais do que em outras áreas da informática, possui um caráter muito prático. Por esta razão, os artigos que exploram os diversos aspectos da disciplina são em geral classificados de especulativos e práticos. Os de caráter especulativo são em geral escritos por profissionais que se abstraem dos aspectos práticos do problema e muitas vêzes propoem soluções julgadas inviáveis de serem utilizadas na prática. Por outro lado, os artigos de caráter puramente prático não raramente carregam em suas propostas os vícios adquiridos no exercício da atividade e não conseguem vislumbrar soluções alternativas fora do modo de proceder do autor. Este artigo, embora possua o caráter especulativo, é baseado em algumas experiências práticas vividas pelo autor.

4.1 - Conscientização do usuário e da equipe de desenvolvimento

Antes de iniciar um projeto baseado no enfoque de versões sucessivas é necessário que tanto o usuário quanto a equipe de desenvolvimento estejam conscientes das características deste enfoque.

Entre as equipes de desenvolvimento é comum encontrar-se pessoas que não tendo entendido o enfoque da prototipagem, acabam por transformá-lo em prototipagem lenta, com todas as desvantagens deste enfoque sem necessariamente as vantagens.

O projetista deve estar consciente, por exemplo, de que os aspectos de eficiência são menos importantes nas primeiras versões para evitar que seja dispendido um tempo excessivo na procura de soluções para problemas ainda não totalmente definidos.

Ao usuário é preciso esclarecer como se dará o desenvolvimento para evitar uma expectativa maior do que a desejável com relação as novas versões. é esperado que a inclusão dos novos requerimentos torne cada vez mais demorada a geração de novas versões.

é preciso, sobretudo, induzir o usuário a participar efetivamente no processo de desenvolvimento, de modo a forçar o uso das primeiras versões do protótipo e da produção de comentários e sugestões que irão influenciar as novas versões.

Deve ser estabelecido um tempo mínimo (1 a 2 meses, dependendo da aplicação) entre a implantação de novas versões, a fim de forçar que os pedidos de modificações sejam bem substanciados e baseados no uso efetivo do protótipo.

4.2 - Características da aplicação

Não é todo tipo de aplicação que sugere o uso do enfoque da prototipagem. ésta equivocado aquele que imagina que o enfoque irá produzir soluções para todos os tipos de problemas.

O enfoque deve ser utilizado principalmente quando não hà maior conhecimento sobre o conjunto de requerimentos que definirão o sistema a ser projetado. Isto pode ser decorrencia da pouca experiência do usuário em sistemas automatizados, ou do pioneirismo do sistema.

O enfoque não deve ser utilizado em situações conhecidas, para as quais, por exemplo, já existem soluções bem difundidas. Não é aconselhável também, a utilização do enfoque em problemas muito grandes, a não ser na primeira ou segunda modalidades, pois o processo evolutivo pode causar muitos problemas quando se começa de uma versão muito reduzida do sistema para se alcançar a solução global. Nestes casos é aconselhável a adoção do enfoque tradicional baseado no ciclo de vida em fases.

4.3 - Características das ferramentas

A primeira característica importante do enfoque da Prototipagem Rápida é que este seja capaz, como o próprio nome sugere, de produzir soluções rápidas, embora parciais e nem sempre as melhores, para os problemas.

Para que isto seja possível os modelos que compõem a ferramenta devem funcionar quase que como um gerador de aplicações onde alguns componentes como relatórios e telas de entrada e saída são gerados a partir de especificações simples fornecidas pelo projetista. Para isto, a linguagem de definição destes componentes deve ser preferencialmente declarativa, não só para facilitar a documentação mas também para possibilitar uma definição mais rápida.

Com relação a estrutura de dados de suporte ao armazenamento das informações (o banco de dados), esta deverá ser capaz de armazenar informações semanticas de modo a facilitar a interação com o usuário na definição dos dados.

Tanto a facilidade de programação, quanto as facilidades para uma organização de arquivos eficiente devem ser proteladas para as fases finais (últimas versões) do processo quando os requerimentos da aplicação estarão mais estabilizados. Tanto a programação como a organização mais adequada dos arquivos visa o aumento da eficiência do sistema, mas consomem um tempo que pode ser totalmente perdido no caso de mudança de requerimentos que afetem a solução. Além disso, a programação dificulta a geração de novas versões.

Obviamente, para se evitar o uso de programação a ferramenta deve incorporar facilidades que permitam este procedimento. Como isto nem sempre é verdade, a programação deve se restringir a umas poucas rotinas onde esta é essencial.

A ferramenta deve possuir um suporte de boa qualidade para documentação para permitir que todos os aspectos que orientaram as decisões de projeto fiquem bem registradas e acessível a todos os envolvidos: projetistas e usuário. É uma boa idéia permitir a documentação "on-line" para permitir que sugestões e reclamações sejam incluidos durante o uso do sistema, assim que estes percebam algum problema na sua utilização. Isto irá simplificar o processo de comunicação entre a equipe de desenvolvimento e os usuários.

O banco de dados deve funcionar também como um repositório das decisões que nortearam a definição das telas de entrada, relatórios, etc, além, é claro, dos dados propriamente ditos. Isto irá possibilitar que o acesso a estas informações se dê de uma maneira uniforme.

Facilidades para auxílio ao uso do sistema (help) devem ser providas pela ferramenta, tanto com relação aos comandos como também com relação aos dados e sua estrutura de armazenamento. Isto possibilitará que o treinamento do usuário seja simplificado já que este contará com um bom esquema de suporte por parte do próprio sistema

5 - CONCLUSÕES

Neste artigo foram sugeridas algumas diretrizes sobre a condução de um projeto baseado no enfoque da prototipagem rápida na terceira modalidade, qual seja, o desenvolvimento do sistema através da geração de versões sucessivas.

Este conjunto de diretrizes visam a tornar mais viável o uso do enfoque no processo de desenvolvimento de aplicações e nortear a definição de uma metodologia baseado neste enfoque.

6 - BIBLIOGRAFIA

[1] Carvalho, A.S.L. Quem desemvolve esta aplicação: O usuario ou o departamento de informatica? Anais do XIX Congresso Nacional de Informatica, pp. 231-237, Rio de Janeiro, 1986.

[2] Silva Neto, A.M. et al. Ambiente de desenvolvimento automatizado. Anais do XIX Congresso Nacional de Informatica, pp. 239-244, Rio de Janeiro, 1986.

[3] Medeiros, G. & Maia, L.C.L. Prototipação de aplicação - Presente ou futuro do desenvolvimento de sistemas de informação? Anais do XIX Congresso Nacional de Informatica, pp. 291-296, Rio de Janeiro, 1986.

[4] Blum, B.I. Still more about rapid prototyping. ACM SIGSOEI Software Engineering Notes Vol 8 No 3, July 1983, pp. 9-11.

[5] Glass, R.L. Recommended: A minimum standard software toolset. ACM SIGSOEI Software Engineering Notes Vol 7 No 4, October 1982, pp. 3-13.

[6] McCracken, D.D. & Jackson M.A. Life cycle considered harmful. ACM SIGSOEI Software Engineering Notes Vol 7 No 2, April 1982, pp. 29-32.

[7] Gladden, G.R. Stop the life-cycle, I want to get off. ACM SIGSOEI Software Engineering Notes Vol 7 No 2, April 1982, pp. 35-39.

[8] Hall, P.A.V. In defence of life cycles. ACM SIGSOEI Software Engineering Notes Vol 7 No 3, July 1982, pp. 23.

[9] Blum, B.I. Rapid prototyping of information management systems. ACM SIGSOEI Software Engineering Notes Vol 7 No 5, December 1982, pp. 35-38.

[10] Keus, H.E. Prototyping: A more reasonable approach to system development. ACM SIGSOEI Software Engineering Notes Vol 7 No 5, December 1982, pp. 94-95.

[11] Taylor, T. & Standish, T.A. Initial thoughts on rapid prototyping techniques. ACM SIGSOFT Software Engineering Notes Vol 7 No 5, December 1982, pp. 160-166.

[12] Mason, R.E.A. & Carey, T.T. Prototyping interactive information systems. Communications of ACM Vol 26 No 5 May 1983, pp. 347-354.

[13] Gomma, H. The impact of rapid prototyping on specifying user requirements. ACM SIGSOFT Software Engineering Notes Vol 8 No 2, April 1983, pp. 17-28.

[14] Zahniser, R.A. Levels of abstraction in the system life cycle. ACM SIGSOFT Software Engineering Notes Vol 8 No 1, January 1983, pp. 6-12.

[15] Lawrence, J.L. Why software is always late? ACM SIGSOFT Software Engineering Notes Vol 10 No 1, January 1985, pp. 19-30.

[16] Alavi, M. An assesment of the prototyping approach to information system development. Communications of ACM Vol 27 No 6 June 1984, pp. 556-563.

[17] Blum, B. Interactive Development of Information Systems: A Case Sudy. Software Practice & Experience, Vol 6 No 6, June 1986, pp. 503-515.

[18] Gomaa, H. The impact of rapid prototyping on specifying user requirements. ACM Software Engineering Notes Vol 8 No 2, April 1983, pp. 17-21.

EITIS:Um ambiente de ferramentas integradas para desenvolvimento de sistemas interativos

Rubens N. Melo
Depto de Informatica PUC-RJ
R. Marques de S. Vicente 225-GAVEA-Rio de Janeiro-BRASIL

EITIS (Environment of Integrated Tools for Interactive Systems) e' um ambiente (ajustavel) para especificar, projetar,implementar e executar Sistemas Interativos de Banco de Dados com Interface de Usuario Grafica. Na realidade o EITIS comporta dois niveis de ambientes: no primeiro nivel (Meta-ambiente) e' definida a meta-informacao necessaria 'as transformacoes das descricoes dos sistemas em desenvolvimento. No segundo nivel essas transformacoes sao efetuadas guiadas pela meta-informacao especificadas no primeiro nivel. O projeto EITIS de certa forma integra uma serie de tecnicas e metodos de varias areas da tecnologia de Informatica (tais como Banco de Dados, Computacao Grafica Interativa e Engenharia de Software) ja' previamente desenvolvidas em projetos anteriores. Este artigo descreve brevemente a concepcao e as principais caracteristicas e decisoes de projeto adotadas no EITIS como uma evolucao de projetos precursores.

1) INTRODUCAO

EITIS (Environment of Integrated Tools for Interactive Systems) e' um ambiente (ajustavel) para especificar, projetar, implementar e executar interativamente Sistemas Interativos de Banco de Dados Nao Convencionais tais como Sistemas de CAD, de Office Automation e Sistemas de BD Graficos.

EITIS, que soa como 80's (dos anos 80 em ingles) e' uma evolucao de varios outras pesquisas desenvolvidas no departamento de Informatica da PUC-RJ desde meados da decada de 70. Ele herdou desses projetos anteriores algumas de suas caracteristicas essenciais:

a)Geracao semi-automatica de sistemas.

A ideia de geracao semi-automatica de sistema nos projetos de pesquisas precursores do EITIS existe desde os tempos de estudos sobre o sistema PSL/PSA da Universidade de Michigan que havia na PUC-RJ desde meados de 74.

Ja' em 1976 com um Processador de Macros desenvolvido para o projeto do mini-computador nacional G-10 [MELO76a], houve a tentativa em alguns trabalhos [BRAN77,ROME83] de gerar programas Fortran correspondentes as descricoes de processos em PSL. Em PSL, os processos (PROCESS) sao descritos como um comentario (COMMENT) e a ideia dessa pesquisa era usar na descricao dos processos um pseudo-codigo com macros possivel de ser traduzido para FORTRAN estruturado (FORTS-PUC) [MELO78a]. Mais tarde um dos projetos de pesquisa (SIMPLES : Um sistema para Implementar Sistemas) [MELO83] tentava gerar a partir de uma especificacao baseada em Entidades e Relacionamentos, o BD descrito em DDL (data definition language) e os programas em Fortran+DML (data manipulation language) do DBTGinho (Um DBMS tipo CODASYL desenvolvido em 75 e utilizado nos projetos e cursos de BD do departamento).

b) Meta descricoes.

Meta-descricoes sao fundamentais para permitir que os sistemas sejam flexiveis e genericos. Esta tem sido uma caracteristica marcante nos projetos precursores do EITIS.

No projeto MIDAS [MELO79a,c,CAMP81,ARAU81] que visava desenvolver um SGBD tipo rede segundo a proposta da CODASYL versao 78 ,com os tres niveis de descricao do Banco de Dados: externo, conceitual e interno, o conceito de meta esquema foi implementado. O esquema do BD dos usuarios era visto tambem como um BD segundo o mesmo modelo (de redes) e o seu esquema era o meta esquema. Desse modo o proprio SGBD era visto tambem como um sistema de BD , um "meta sistema de BD".

No projeto do SDDPUC [LOBE81a,b,MELO81a,ALVA82]o conceito de meta dicionario foi introduzido e implementado. O SDDPUC era um "sistema de dicionario de dados flexivel" que permitia o usuario "autorizar" (verbo estranho que foi usado na implementacao do SDDPUC) a linguagem a ser utilizada na descricao do sistema. Desse modo era possivel descrever o sistema em questao em diferentes niveis . Por exemplo podia-se descrever a visao conceitual ("gerencial") ou a visao operacional (de implementacao) do sistema bastando para isso mudar ("autorizar") no meta dicionario os conceitos a utilizar (os objetos, relacionamentos e atributos).

No SDDPUC O usuario interagia com linguagem ("natural") de formato livre ou atraves de uma interfaces de menus [OPPE82]. Um subsistema de verificacao da consistencia da especificacao foi projetado em [deMELO82]. As linguagens de especificacao geradas mantinham a mesma estrutura, somente os termos, os verbos, os atributos mudavam. O meta-modelo era o "Entidades-Relacionamento-Atributos" (modelo E-R) [CHEN76].

c) Tipos abstratos e Modelos Conceituais de BD

Em todos os projetos anteriores ao EITIS as ferramentas e sistemas desenvolvidos eram orientados para Sistemas de Informacao baseados em Banco de Dados[GUSM77].

Desde muito tempo atras o enfoque "infologico" para sistemas de informacao [LANG66,SUND73,GUSM77,TEIX79,ANDR82b] preconizava a descricao conceitual de Sistemas de BD. A partir da proposta da ANSI/SPARC [ANSI75] varios modelos conceituais ou semanticos tem sido propostos na literatura, sendo os modelos baseados em Entidades e Relacionamentos os mais aplicados em geral quer em sistemas de BD ou convencionais. O enfoque de E-R combinado com metodologia de analise e projeto estruturado de sistemas [GANE83] foi introdusido em [SOUZ81]

Ha' alguns anos varios estudos comparativos de modelos semanticos tem sido feitos [PENA83,MELO84b,ZAKI85]. Dentre esses modelos o modelo E-R e suas extensoes tem tido papel importante nos projetos anteriores ao EITIS

O uso do modelo E-R e suas extensoes como modelo para especificacao de sistema foi utilizado em varios projetos anteriores [MELO77,MELO79b,ANDR82b] e como modelo de SGBD foi utilizado no projeto REDAS: "Relation-Entity Data System" [GOME77,MELO78b]. Mais tarde um outro projeto de pesquisa o REDAS II, incluia extensoes ao modelo E-R e tecnicas de Tipos abstratos.

O uso de Tipos abstratos em Banco de Dados e' ideia antiga [MEL076b] e como extensao de BD Relacional para aplicacoes nao convencionais foi utilizado no projeto STAD: "Sistema de Tipos Abstratos de Dados" [NIGR84,MEL084a,BARB85] onde o conceito de objeto complexo em BD para CAD ja' era abordado.

d) Computacao Grafica Interativa e Interfaces de Usuario

Aproveitando a experiencia anterior com computacao grafica em projetos antigos [MEL072a,b,c,d] e com novas possibilidades de recursos graficos (estacoes graficas, micros, etc...) o uso de tecnicas graficas para as interfaces de usuarios se intensificou no projeto EITIS.

Implementacoes de pacotes graficos padronizados tipo CORE e GKS [ALVA84,CAVA85,TANA85a,b] e de editores Graficos [CAVA85] e de gerenciadores de janelas sao contribuicoes recentes para o projeto EITIS.

Interfaces amigaveis para interacao com Banco de Dados foram projetadas e implementadas em sistemas anteriores [OPPE82,THOM84]. O conceito de SGIU (Sistema de Gerencia de Interface de Usuario) que generaliza o conceito de Interface de Usuario foi inicialmente estudado em [FUKS84] e em [THOM84] e tem papel muito importante no EITIS.

2)O PROJETO EITIS

A juncao do conceito de SGIU com o conceito de meta descricoes do MIDAS e SDDPUC, com as ideias de geracao semi-automatica de sistemas dos projetos antigos e os desafios modernos das novas areas de aplicacao de BD, tais como CAD [CABR85,MEL085b], Office Automation e outras [LANZ86,LANZ87], deram inicio ao projeto EITIS. A ideia basica era integrar uma variedade de ferramentas de tal forma que fosse possivel configurar varios ambientes. Por exemplo, um ambiente para desenvolver sistemas de CAD, outro voltado para sistemas de SAD (Sistema de Suporte a Decisoes), outro para sistemas de BD Graficos, etc.. A fig. 1 mostra os principais componentes do ambiente EITIS.

Um ambiente EITIS considera que todo sistema a desenvolver tem tres aspectos fundamentais : Dados , Processos e Controle. O processo de desenvolvimento segue o metodo de transformacoes de descricoes do sistema em varios niveis de abstracao [BATI86,MEL086]. Em cada nivel ha' ferramentas para descricao dos tres aspectos dos sistemas e tambem para a verificacao. Cada transformacao e' guiada pelas descricoes anteriores e por meta-informacoes inicialmente geradas previamente (e possivelmente atualizadas) pelos operadores das transformacoes (especificadores do nivel conceitual, projetistas, usuarios finais, etc)

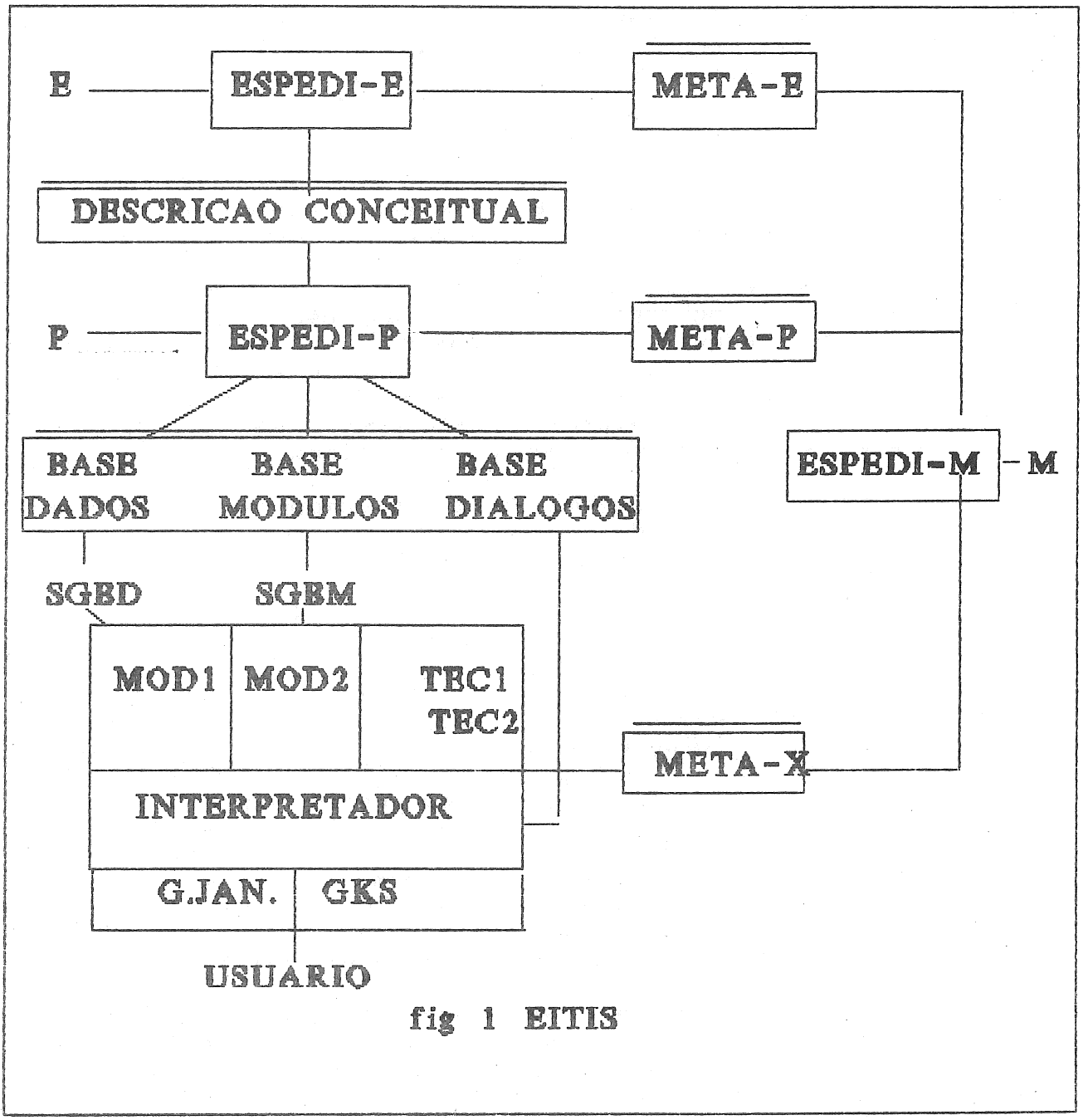

E
ESPEDI-E
META-E
DESCRICAO CONCEITUAL
P
ESPEDI-P
META-P
BASE DADOS
BASE MODULOS
BASE DIALOGOS
ESPEDI-M
M
SGBD
SGBM
MOD1
MOD2
TEC1
TEC2
META-X
INTERPRETADOR
G.JAN.
GKS
USUARIO

fig 1 EITIS

O projeto EITIS considera dois niveis de ambientes: O meta ambiente que "autoriza" , isto e' ,define as meta-informacoes do ambiente tais como as linguagens dos usuarios especificadores, regras de mapeamento para os projetistas, perfis de usuarios iniciais na execucao etc.

No nivel de especificacao o subsistema ESPEDI-M [SILV85,MELO85a] permite a definicao do meta dicionario contendo os termos e verbos a utilizar na especificacao (por exemplo ESTADO, TRANSICAO, ACIONA MODULO, ENTIDADE, ATRIBUTO, ...)

O especificador usando essa linguagem especifica atraves do subsistema ESPEDI-E o sistema a ser gerado. Nessa especificacao pode-se descobrir os tres aspectos do sistema: Os dados da aplicacao, os processamentos e a dinamica de acionamento desses processamentos (a interface do usuario). Nesse nivel a linguagem de descricao usa algum modelo conceitual de dados(ER por exemplo), algum modelo de descricao de processos (DFD por exemplo) e algum modelo de descricao de dialogos (ATN - Augmented Transition Network - por exemplo). Os dialogos descrevem como o usuario controlara' a ativacao dos modulos do sistema.

Uma parte do subsistema ESPEDI-E comporta funcoes que permitem o especificador analizar a descricao feita e verificar sua consistencia e completeza.

Os tres aspectos do sistema sao mapeados em todos os niveis.

No nivel de projeto de implementacao o ESPEDI-M define regras para o mapeamento da especificacao em estruturas logicas de dados (Relacional por ex.), de processamento (DEM por ex.) e de interface de usuario (Dialogos). O projetista atraves do ESPEDI-P sera' ajudado por estas regras (META-P) para efetuar a transformacao da descricao conceitual em estruturas de implementacao do sistema. Uma parte do ESPEDI-P comporta funcoes que permitem o projetista verificar a consistencia do projeto. A transformacao da descricao de dialogos representados por ATNs em estruturas de dados de um BD tipo rede foi estudada em [GUER86].

No nivel de execucao o meta ambiente orienta em termos de que conjunto de tecnicas de interacao sao apropriadas aos usuarios (estilos de usuarios). Esses estilos podem ser iniciais e serem atualizados durante a interacao do usuario com o sistema.

No nivel de execucao o usuario interage com o sistema atraves de facilidades graficas (Janelas,menus,etc) de acordo com o estilo determinado por seu perfil [FUKS84,BATI86,MELO86]. O acionamento de modulos ocorre durante a interpretacao do dialogo. Os modulos sao acessados

na base de modulos via facilidades de Gerencia de Base de Modulos de maneira similar ao acesso aos dados do BD atraves de Gerenciador de BD.

As meta-informacoes utilizadas pelos ambientes EITIS podem ser vistas como "bases de conhecimento" das metodologias empregadas. Por exemplo na analise de consistencia, feita pelo ESPEDI-E, da descricao conceitual ele se vale da meta-informacao (base de conhecimento) que define as regras de consistencia e completeza (Um relacionamento incompleto, um processo sem saidas , etc). Da mesma maneira no mapeamento do nivel conceitual para o de implementacao as regras de transformacao de relacionamentos em tabelas, ou criterios para definicao de indices secundarios formam uma base de conhecimento sobre esse metodo de implementar sistemas.

3)A EVOLUCAO DO PROJETO EITIS

Basicamente o meta ambiente sugerido pelo EITIS e' do tipo "configurativo interativo" que pode ser (no futuro) tambem orientado por assistentes especialistas que usam bases de conhecimentos sobre os ambientes objetos desejados e sobre metodologias adequadas.

Atualmente os ambientes EITIS seguem o paradigma de processo de projeto em fases (especificacao conceitual, projeto de implementacao, execucao) com transformacoes semi-automaticas entre as descricoes ajudadas por base de conhecimento sobre os mapeamentos. Espera-se que interativamente (com o ESPEDI-M) possa-se escolher os modelos e ferramentas apropriados em cada nivel

Varias topicos estao sendo analizados dentro do ambiente EITIS visando sua generalizacao de aplicacao. Por exemplo:

3.1) Generalizacao do ESPEDI-M

A busca de um Meta Modelo dos Ambientes gerados e' fundamental para que se possa ter como instancias de ambientes gerados, ambientes variados tais como de CAD,CASE,CAI,SAD,etc.

Visando a generalizacao de aplicacao as ferramentas sendo desenvolvidas sao alternativas para configuracao de diferentes ambientes. Por exemplo mais de um DBMS , diferentes gerenciadores de janelas, diferentes interpretadores universais de dialogos, diferentes nucleos de rotinas graficas, etc....

Com a ideia de configuracao interativa inteligente, o ESPEDI-M conversaria com o "meta-projetista" e utilizando uma base de conhecimento de ambientes, os dois

configurariam um ambiente desejado a partir de varias ferramentas existentes, por exemplo um DBMS Relacional + um Interpretador de dialogos em ATN + um Gerenciador de Janelas superpostas seriam configurados num ambiente para sistemas de CAD e assim por diante.

3.2) A integracao via base de dados orientada p/ objetos

A orientacao para Objetos tem sido considerado um possivel paradigma integrador de varias areas como as de Inteligencia Artificial, Linguagens de Programacao e Banco de Dados [MEL087].

No EITIS tanto os dados no BD como os modulos na Base de Modulos como os Dialogos sao objetos que devem ser manipulados de maneira similar na execucao. Estes objetos tem caracteristicas tais como: versoes (em tempo), alternativas (de projeto), etc... que seriam melhor tratadas com um Gerenciador de Banco de Dados Orientado para Objeto.

Estudos de modelagem orientada para objetos dos tres aspectos de sistemas nos ambientes EITIS estao em andamento visando favorecer a integracao desejada entre as ferramentas componentes.

4) CONCLUSOES

A busca de ambientes integrados para desenvolvimento de software tem se tornado uma das areas de pesquisa mais importantes em Engenharia de Software.

Neste artigo foi feito uma breve descricao do EITIS, um ambiente que esta sendo desenvolvido no departamento de Informatica da PUC-RJ. Nessa descricao foi mostrado como este ambiente, que incorpora varios aspectos importantes da tecnologia atual em Engenharia de Software, evoluiu naturalmente de outros projetos de pesquisa anteriores.

Referencias

[MELO72a] Melo,R.N. "Um pacote de subrotinas para uso do Display IBM 2250 em Fortran",Div. de Comp,ITA,1972

[MELO72b] Melo,R.N. "Desenho animado no Display IBM 2250", Div. Comp.,ITA,1972

[MELO73c] Melo,R,N. "Corte e aterro de pistas de aeroportos interagindo com um display IBM 2250", Div. Comp., ITA, 1972

[MELO72d] Melo,R.N. "Representacao e manipulacao de superficies x(u,v), y(u,v) e z(u,v) no display IBM 2250",Div. Comp. ITA, 1972

[MELO76a] Melo,R.N.,Schwabe,D. "Extending the Control Structures of FORTRAN via Macro Generator",Tecnical Report n.1/1976, PUC-RJ

[MELO76b] Melo,R.N. "Towards a DBMS based on Abstract Data Types" in Processings of "Simposio Internacional sobre Metodologias para Projeto e Construcao de Sistemas de Software e Hardware" Rio de Janeiro 1976

[MELO77a] Melo,R.N.,Lellis,L. "O Mapeamento da Especificacao Conceitual em Banco de Dados Gerenciado pelo TOTAL",Teccnical Report n.14/1977 PUC-RJ

[MELO78a] Melo,R.N.,Steinbruck, P."FORTS - Um Preprocessador para Programacao Estruturada em FORTRAN" Tecnical Report n. 19/1978 - PUC-RJ

[MELO78b] Melo,R.N.,Gomes, A. F., Silva, C. S."REDAS: Um Sistema de Gerencia de Banco de Dados Baseado no Modelo de Entidades e Relacoes" in Proc of "IV Seminario Integrado de Hardware e Software", Rio de Janeiro - 1978

[MELO79a] Melo,R.N.,Gomes, A.F."Aspectos Significativos da Especificacao de um Sistema de Gerencia de Banco de Dados tipo DBTG" In Proc of EXPODATA 79,Valparaiso,Chile, 1979

[MELO79b] Melo,R.N.,Lellis, L."O Mapeamento da Especificacao Conceitual em Esquema Interno Gerenciado pelo ADABAS in Proc of XII congresso Nacional de Processamento de Dados" - Sao Paulo - SP - 1979

[MELO79c] Melo,R.N. "The Monitoring of Integrity Constraints in a CODASYL like DBMS in Proc of V International Conference on Very Large Data Base" Rio de Janeiro - RJ - 1979

[TEIX79] Teixeira, C.,Melo,R.,N. "An Experience with the ANSI/SPARC approach to Data Base Design" in Special Issue on Work in Progress in Developing Countries"of "V Int Conf on Very Large Data Base" RJ -1979

[MELO81a] Melo,R.N.,Lobel, E.R. "SDDPUC : Um Sistema de Dicionario de Dados Flexivel"- Congresso Nacinal de Processamento de Dados SUCESU - 1981

[LOBE81a] Lobel, E.R.,Melo,R.N. "Um Estudo sobre Dicionario de dados" Tec. Rep PUC-RJ Rio de Janeiro - RJ - 1981

[ANDR82a] Andrade, A.D., Veras, P.,Melo,R.N. "Um Estudo sobre Metodologia de Programacao" Tec. Rep. PUC-RJ - Rio de Janeiro -1982

[ALVA82] Alvarenga,C.,D'Ipolito,C.,Melo,R.N. "Manual do Usuario do SDDPUC" - Tec. Rep. PUC-RJ Rio de Janeiro - 1982

[MELO83] Melo,R.N.,Silva, J.G., Araujo, F.S.S., "SIBER - Sistema Integrado Baseado em Entidades e Relacionamentos" - XIV Congresso Nacional de Informatica - SUCESSU Sao Paulo - SP - 1983

[ALVA84] Alvarenga,C.,Tanaka,a.,Melo,R.N., "Padronizacao em Sistemas Graficos" Tec Rep Depto Inform. Puc 84

[MELO84a] Melo,R.N.,Nigri,V. "STAD Uma Interface de Tipos Abstratos de Dados para BD para CAD" SUCESU 84

[MELO84b] Melo,R.N.,Zakimi,B.,Cordero,J. "Um estudo comparativo deModelos Semanticos de BD" " RBC-(Revista Bras. de Comp.) Julho-84

[ZAKI85] Zakimi,B., Vieira,M.,T.,P.,Melo,R.N.,"Ferramentas Graficas para o Projeto de BD" SUCESU 84 e na Conf. Internacional da Espanha 85

[MELO85a] Melo,R.N.,Silva,D.,S.,"Uma ferramenta para especificar sistemas interativos" SUCESU 85

[MELO85b] Melo,R.N.,Zakimi,B.,Vieira,M.,"SGBDs orientados p/ aplicacoes de CAD" SUCESU (Brasilia 85)

[MELO86] Melo,R.N.,Batista,G.,R.,"Sistemas graficos Interativos:Um ambiente de software e uma metodologia de projeto",SUCESU 86

[TANA85a] Tanaka,A.,K.,Melo,R.N., "Implementacao do GKS-Graphical Kernel System", SBC/Soft. Basico - Belo Horizonte, 1985

[LANZ86] Lanzelloti,R.,Melo,R.N.,"O Uso de Tecnicas de Inteligencia Artificial no Projeto de Ling. de Consulta de BD" SEMISH 86 SUCESU 86

[LANZ87] Lanzelloti,R.,Melo,R.N.,"A base de conhecimento de um otimizador de consultas a BD Relacional",SEMISH 87

[MELO87] Melo,R.N. "Banco de Dados Nao Convencionais". VI JAI, VII Congresso SBC, Salvador 1987

[GANE83] Gane,C.,Sarson,I."Analise Estruturada de Sistema",LTC,1983

[LANG66] Langfors,B."Theoretical Analysis of Information Systems",Student-litteratur and Aucherbach,Sweden,1966

[SUND73] Sundgren,B."An infological approach to Data Bases".Phd Thesis,University of Stockholm,1973

[CHEN76] Chen,p.p."The Entity-Relationship model:Towards a unified view of data".ACM TODS v.1, March 1976

[ANSI75] ANSI/X#/SPARC Study Group on DBMS. Interim Report. FDT(ACM SIGMOD) n.2 1975

[GUSM77] Gusmao,E.,D."Especificacao Conceitual de Sistemas de Informacao Apoiados em Banco de Dados".Tese de mestrado, Depto Inf PUC-RJ.1977

[BRAN78] Brandao,L.,C."Ferramentas para a Implementacao de Sistemas de Informacao Apoiados em Banco de Dados".Tese de mestrado. Depto Inf PUC-RJ.1978

[ANDR82b] Andrade,D.A."Uma Sistematica para Projeto de Sistemas de Banco de Dados".Tese de mestrado.Depto de Informatica.PUC-RJ.1982

[ROME83] Romero,M.,A."Geracao Automatica de Programas usando a Metodologia de Jackson".Tese de mestrado. Depto Inf PUC-RJ.1983

[PENA83] Pena,J.,C."Estudo Comparativo dos Modelos de Dados Semanticos".Tese de mestrado.Depto Inf PUC-RJ.1983

[LOBE81b]Lobel,R.,E."Uma Contribuicao para a Especificacao e Implementacao de Sistemas de Dicionario de Dados Flexiveis".Tese de mestrado.Depto Inf PUC-RJ.1981

[SOUZA81] Souza,C,T."O Enfoque de Entidade e Relacionamentos na Analise e Projeto Estruturados de Sistemas".Tese de mestrado.Depto Inf. PUC-RJ.1981

[CAMP81] Campos,R.L."Um Subsistema de Definicao de Dados para um SGBD CODASYL-78".Tese de mestrado.Depto Inf. PUC-RJ.1981

[ARAU81] Araujo,P.S.S."Um Subsistema de Manipulacao de Dados para um SGBD CODASYL-78".Tese de mestrado.Depto Inf PUC-RJ.1981

[OPEN82] Oppenheimer,M."Uma Facilidade para Usuario Final do SDDPUC".Tese de mestrado.Depto Inf. 1982.

[deMEL82] de Melo,S.R.V."Uma Facilidade para Verificacao da Especificacao de Sistemas com SDDPUC".Tese de mestrado.Depto de Inf PUC-RJ.1982

[FUKS84] Fuks,H."Sistemas de Gerencia de Interface de Usuario ".Tese de mestrado.Depto Inf. PUC-RJ.1984

[THOM84] Thome,L.F."Sistemas Interativos de Banco de Dados".Tese de mestrado.Depto Inf. PUC-RJ.1984

[NIGR84] Nigri,V."STAD Uma interface de Tipos Abstratos de Dados para BD p/ CAD".Tese de mestrado.Depto Inf. PUC-RJ

[TANA85b] Tanaka,A."Sistema grafico de apoio a projetos de engenharia".tese de mestrado. Instituto Militar de Engenharia (IME).1985

[SILV85] Silva,S.D."Uma ferramenta para especificacao de Sistemas Interativos".Tese de mestrado.Depto Inf.PUC-RJ.1985

[CAVA85] Cavalcanti,P.R."Ferramentas para Modelagem Geometrica e Edicao Grafica".Tese de mestrado.Depto Inf. PUC-RJ

[CABR85] Cabral,J."Extensao de Sistemas de Gerencia de Banco de Dados para CAD".Tese de mestrado.Depto Inf. PUC-RJ

[BARB85] Barbosa,A."Facilidades de BD Relacional para aplicacoes nao Convencionais".Tese de mestrado.Depto Inf. PUC-RJ

[BATI86] Batista,G.R.,"Metodologia de projeto de Dialogos em Sistemas Graficos Interativos".Tese de mestrado.Depto Inf. PUC-RJ

[GUER86] Guerreiro,A." SGIU : A Geracao e Interpretacao de Dialogos".Tese de mestrado.Depto Inf. PUC-RJ

[GOME77] Gomes,A.F."O modelo de dados do sistem REDAS".Tese de mestrado.ITA,1977

DISEÑO, ESPECIFICACION Y PROTOTIPIFICACION DE SISTEMAS INTERACTIVOS

Víctor Manuel Toro C.

Grupo "ISAC: Ingeniería de Software Apoyada por Computador"
Departamento de Sistemas y Computación
Universidad de los Andes
Apartado Aéreo 4976
Bogotá - Colombia

"XIII Conferencia Latinoamericana de Informática"
Bogotá, Noviembre'87

RESUMEN: Para describir globalmente un sistema de software es imprescindible disponer de un "lenguaje" capaz de expresar de manera general (¡pero concreta!) los principales aspectos del sistema. La escogencia de este lenguaje es particularmente crucial para la labor de Diseño, puesto que esta etapa tiene justamente el objetivo de describir sistemas de software que aún no existen. Un buen lenguaje de diseño debe ser general, con una clara interpretación intuitiva, y con un sólido respaldo formal. Pero además, debe ser susceptible de procesamiento automático, de manera tal que se pueda avanzar fácilmente hacia la siguiente etapa de desarrollo: la programación. Este artículo presenta una metodología (i.e. herramientas + criterios) clara y efectiva para diseñar sistemas interactivos de complejidad mediana. Se explora asímismo la posibilidad de procesar automáticamente dichos diseños, con miras a obtener directamente prototipos del sistema en construcción.

0. INTRODUCCION

"... hay que terminar rápidamente la etapa de diseño, para que al final del proyecto quede tiempo suficiente para poder resolver los problemas causados por haber hecho muy rápido la etapa de diseño ..." [MYER 78].

La Ingeniería de Software está cambiando radicalmente. El clásico Ciclo de Vida del Software, sobre el cual se apoyan numerosos conceptos y métodos, está dejando de ser adecuado. En su lugar están apareciendo diversas propuestas: prototipificación, especificación operacional, desarrollo transformacional, metodología Jackson, ... [AGRE 86].

Uno de los aspectos más polémicos es el Diseño. En realidad, desde hace bastantes años existen métodos y lenguajes para diseñar (i.e. describir globalmente antes de desarrollar) sistemas de software comercial, que tradicionalmente daban lugar a aplicaciones tipo "batch" [YOUR 76]. Sin embargo, estos métodos y lenguajes de diseño empezaron a mostrarse inadecuados en muchos casos, y se fueron convirtiendo más en un obstáculo que en una ayuda [YOUR 86]. Como consecuencia, en los últimos años han sido identificadas al menos cuatro categorías diferentes de diseño [PRES 87]:

- diseño orientado por flujo de datos
- diseño orientado por objetos
- diseño orientado por estructuras de datos
- diseño de sistemas de tiempo real.

En este artículo se presenta una metodología apropiada para diseñar, especificar y prototipificar sistemas interactivos. Esta metodología, que puede ser clasificada en la línea de diseño orientado por objetos, fué afinada a lo largo de un proyecto culminado recientemente, en el cual se construyó un ambiente de aprendizaje de la programación [LOPE 87a]. Adicionalmente, ha venido siendo probada en varios cursos-proyecto de pregrado y postgrado, en general con muy buenos resultados. En este artículo se tratarán los siguientes temas:

1. Caracterización de los Sistemas Interactivos
2. Los Diagramas de Transiciones
3. Especificación detallada de Sistemas Interactivos.
4. "Algoritmo" para diseñar Sistemas Interactivos
5. Compilación de Diagramas Estructurados de Transiciones
6. Prototipificación de Sistemas Interactivos
7. Conclusiones

Es de anotar que algunos ejemplos y analogías presentados en esta trabajo harán referencia al conocido editor WordStar®. Esto se hace únicamente para facilitar la explicación de ciertos conceptos, pero no implica en ningún momento que lo aquí expuesto solo tenga aplicación a los problemas de edición de texto.

1. CARACTERIZACION DE LOS SISTEMAS INTERACTIVOS

Un Sistema Interactivo S./I. es un conjunto de servicios y facilidades a disposición del usuario, que funcionan, se activan o suspenden siguiendo las órdenes del usuario. Algunas de las diferencias fundamentales entre los Sistemas Interactivos y los sistemas "batch" clásicos son las siguientes:

- Los S./I. no poseen "entradas" y "salidas" definidas de antemano (p. ej.: relación de horas extras, novedades de descuentos ➡ Cheques de nómina, relación de retenciones). En los S./I. la entrada es una sucesión de datos y comandos seleccionados directamente y sobre la marcha por el usuario (p. ej.: letras, números, 'suba renglón', 'borre palabra', 'cambie margen', 'inserte',...) que eventualmente producirán como resultado lo que el usuario quiere obtener.

- Un S./I. no "hace" una determinada labor (p.ej.: procesar la nómina, actualizar el inventario, ...), sino que facilita que el usuario haga su propia labor (p.ej.: preparar un informe, calcular un presupuesto, evaluar un diseño,...).

- Los S./I. no son susceptibles de ser calificados como globalmente correctos o incorrectos. En efecto, la noción de corrección es una relación entre entradas y salidas bien determinadas [GRIE 81][BOHO 86], lo cual no es el caso de los S./I.. La corrección en los sistemas interactivos está confinada a la implementación correcta de cada una de las diversas facilidades y servicios que se ofrecen al usuario.

El objetivo primordial de un S./I. es ser ***adecuado***, es decir, permitir que el usuario pueda realizar efectivamente las funciones que necesita para su labor. En otras palabras, el sistema interactivo debe ofrecer al usuario las facilidades y servicios suficientes para que éste pueda realizar su labor rápida y cómodamente.

Esta es justamente la principal deficiencia de los métodos clásicos de diseño: no considerar ni permitir expresar en forma explícita el papel activo del usuario. Para describir correctamente un sistema interactivo es necesario entonces un lenguaje y un método de diseño en el que:

- aparezca explícito el papel del usuario
- se muestren los servicios que el usuario tiene eventualmente accesibles en cada instante
- se muestre la manera de seleccionar tales servicios
- se pueda especificar rigurosamente la acción que realiza cada uno de dichos servicios

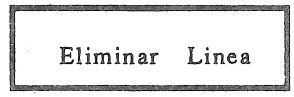
2. LOS DIAGRAMAS DE TRANSICIONES

Para describir Sistemas Interactivos se van a utilizar los "Diagramas de Transiciones". Este formalismo fué sugerido inicialmente por A. I. Wasserman [WASS 81], aunque sin mayor elaboración. En este trabajo se ha refinado dicho formalismo, incorporándole además conceptos modernos como especificación formal [GEHA 86], certificación de la corrección del software [GRIE 81][BOHO 86], y tipos abstractos de datos [CAR86][CAR87].

Un **Diagrama de Transiciones** es un grafo Bi-Partito definido por:

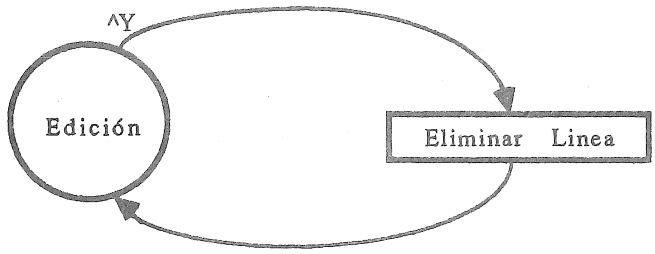
Nodos:

* Conjunto de Niveles p.ej.:

* Conjunto de Procesadores de Comando p.ej: 

Arcos:

- Los arcos van de un "Nivel" a un "Procesador de Comando", o viceversa.
- Los arcos que van de un Nivel a un Procesador de Comando pueden estar marcados con una etiqueta llamada "Selector".
- Se llaman "Transiciones" a los caminos de la forma siguiente:



ó bien

--- * ---

Los Niveles modelan los *estados de interacción*, esto es, los instantes de un sistema interactivo en los cuales el usuario decide cual es la siguiente acción que debe ejecutarse. Más adelante se verá que, desde un punto de vista conceptual, un Nivel no es más que la materialización de un Tipo Abstracto de Datos [CARD 86][CARD 87], es decir, un objeto más un repertorio de acciones o comandos que pueden actuar sobre él.

Los Procesadores de Comando modelan los *estados de ejecución*, esto es, la ejecución de las acciones que ordena el usuario.

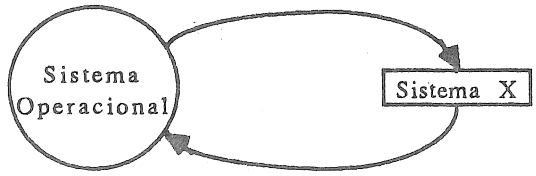
Las Transiciones modelan la secuencia "el usuario puede seleccionar alguna acción", "se realiza la acción seleccionado por el usuario", "el usuario puede seleccionar la siguiente acción".

---*---

De la anterior definición se deduciría que cualquier grafo bi-partito de 'Niveles' y 'Procesadores de Comando' sería un Diagrama de Transiciones válido. Sinembargo, con el fin de poder manipular estos diagramas con herramientas y procedimientos automatizados, es conveniente restringirse a una clase que llamaremos los "Diagramas Estructurados de Transiciones". La práctica muestra que esta restricción no impide expresar complejos sistemas interactivos: todo lo contrario, permite hacerlo con mayor claridad y certeza.

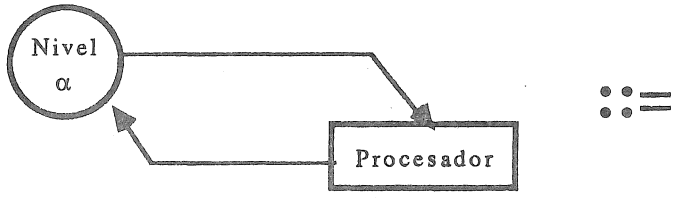
Los **Diagramas Estructurados de Transiciones** son aquellos que se derivan de la siguiente gramática:

Símbolo Distinguido:

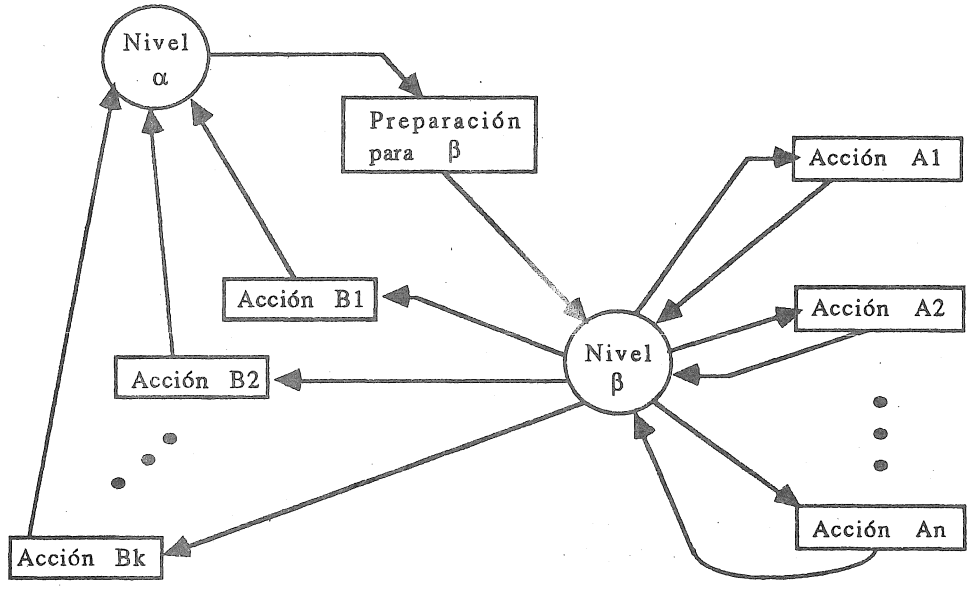


Regla única de Reescritura:

Parte Izquierda:



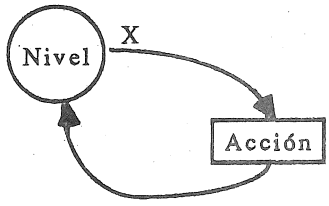
Parte Derecha:



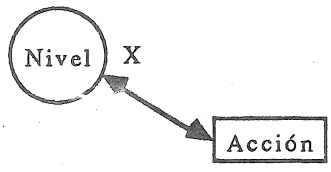
donde:

* k ≥ 1, es decir, existe al menos una transición de regreso al Nivel padre;
* existe una transición A_i ó B_j encargada de procesar el caso de "error", es decir, la selección por parte del usuario de una acción no disponible en ese nivel.

NOTA: En lo sucesivo



se abreviará así



---*---

Como ilustración de las reglas anteriores, a continuación se muestra un Diagrama Estructurado de Transiciones que presenta un subconjunto de WordStar®:

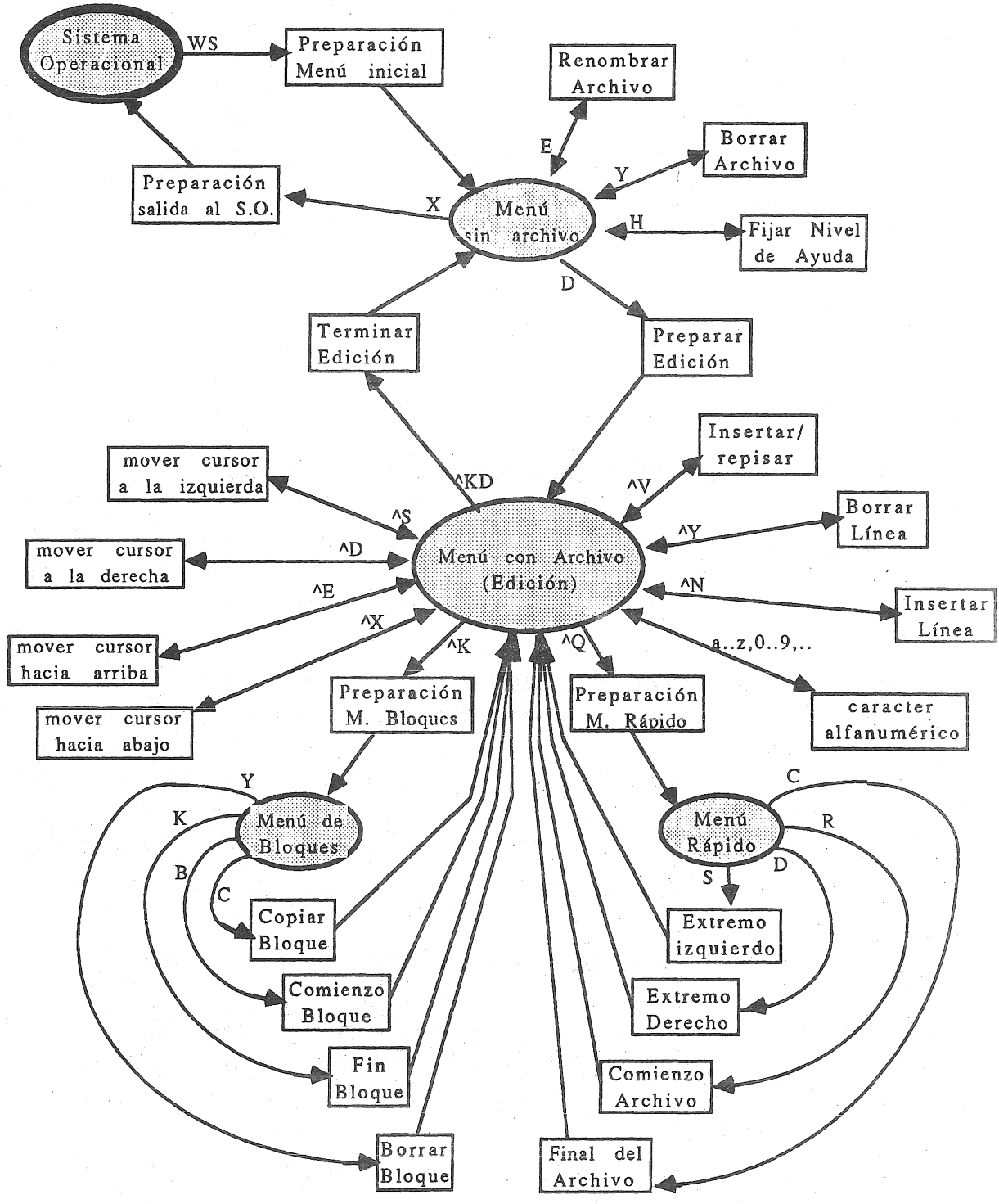


Figura Nº 1: Diagrama de Transiciones de un subconjunto de WordStar®

3. ESPECIFICACION DETALLADA DE SISTEMAS INTERACTIVOS

Los Diagramas de Transiciones permiten describir globalmente un sistema interactivo, es decir, establecer qué servicios y en qué contexto estarán disponibles para el usuario. Sin embargo, para poder obtener un mayor provecho de la labor de diseño, es necesario detallar la forma y apariencia de cada uno de los Niveles, y el efecto de cada uno de los Procesadores de Comando. Esta labor de describir en detalle **qué** hacen las partes de un sistema sin explicitar aún el **cómo** es lo que se conoce con el nombre de "Especificación".

3.1 ESPECIFICACION DE LOS NIVELES

Especificar un Nivel significa establecer claramente:

* **El conjunto de Comandos accesibles**: Es de resaltar que esto no significa que todos los comandos de un nivel estén accesibles en todo momento. Probablemente para poder activar ciertos comandos sea necesario que el usuario haya llevado el sistema a determinadas condiciones (p. ej.: para poder "borrar bloque" se requiere que haya un "bloque marcado").

* **El conjunto de Variables de Estado**: son aquellas que indican la situación de los objetos que se manipulan en dicho Nivel en un momento determinado. Se distinguen porque serán frecuentemente consultadas y/o modificadas por varios de los Procesadores de Comandos que se desprenden del Nivel. Ejemplos de variables de estado de un sistema interactivo de edición de texto podrían ser: 'posición del cursor', 'margen izquierda', 'insertando o repisando', 'total de líneas', ..., y particularmente 'texto en edición'.

  Dos observaciones son importantes con respecto a estas Variables de Estado:

  - Un Nivel hereda las Variables de Estado asociadas a sus Niveles ancestro, es decir, sus Procesadores de Comando pueden consultar y/o modificar las Variables de Estado de los Niveles superiores.

  - La manera más cómoda y eficiente de implementar las Variables de Estado es con variables globales (i.e. accesibles desde cualquier parte del programa). Esto evita colocar siempre los mismos parámetros en muchos procedimientos, lo cual recarga enormemente la sintaxis de un programa.

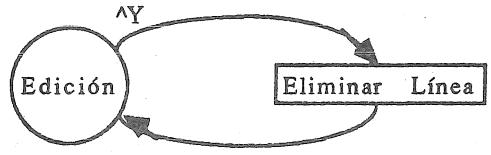
* La Especificación de la Pantalla: Desde un punto de vista meramente computacional, la mayor parte del código de un programa interactivo está dedicado a dibujar, refrescar o actualizar la pantalla, de manera tal que ésta refleje adecuadamente la acción de los comandos que ha dado el usuario. Se requiere entonces especificar con detalle la configuración y disposición de la pantalla en cada uno de los Niveles. Se distinguen cuatro componentes en la especificación de la pantalla asociada a un Nivel:

  - **Parte Fija**: menú, ayudas o mensajes fijos que siempre aparecen en la pantalla;
  - **Parte Variable**: es la imagen de Variables de Estado del Nivel o de Niveles superiores (p.ej.: porción del texto en edición, posición del cursor, posiciones de tabulación, ...);
  - **Parte Entorno**: es la imagen del entorno en el cual se está ejecutando el programa (p.ej.: disco activo, directorio de archivos, tipo de terminal, ...);
  - **Interfase usuario↔ sistema para selección de comandos**: especificación de los mecanismos de que dispone el usuario para seleccionar el comando que desea (p. ej.: selección directa de comandos; preselección, cambios de preselección y confirmación; preselección anticipada del comando más frecuente, etc.).

* Aserciones de Corrección del Nivel: Cada Nivel tiene asociado un conjunto de Variables de Estado (propias y heredadas) y una configuración particular de la pantalla (la cual probablemente incluye la imagen de Variables de Estado). Se pueden entonces establecer Aserciones de Corrección del Nivel, es decir, afirmaciones que caractericen el comportamiento correcto del sistema (p.ej.: "el trozo de texto que aparece en pantalla debe estar alineado según la variable de estado 'margen izquierda'").

3.2 ESPECIFICACION DE LOS PROCESADORES DE COMANDOS

Los Procesadores de Comando deben ser especificados en el contexto de la Transición en la cual se encuentran. Tomando como base de ejemplo la Transición que se muestra a continuación, especificar un Procesador de Comando significa establecer claramente los siguientes aspectos:



* El Nivel de Origen.

* El Nivel de LLegada, eventualmente el mismo de origen.

* La PreCondición: esta condición se construye con la aserción de corrección asociada al Nivel de Origen más un "bautizo" de la situación en la que se encuentran las Variables de Estado que serán afectadas por el Procesador de Comando en cuestión (p.ej. " el sistema está disponible para seguir editando" Y "el texto tiene N líneas" Y "el cursor se encuentra en la línea k del texto" Y "el cursor aparece en la fila f, columna c de la pantalla" ...).

* La PostCondición: en general esta condición tiene la forma de la siguiente conjunción: "aserción de corrección asociada al nivel de destino" Y "se efectuó la acción que había seleccionado el usuario sobre las variables de estado" Y "se reflejó en pantalla dicho cambio en las variables de estado" (p. ej.: "el sistema está disponible para seguir editando" Y "el texto tiene N-1 líneas" Y "el texto ya no tiene la linea k" Y "se suprimió de la pantalla la fila f" Y "el cursor aparece en la fila f, columna 1 de la pantalla" ...).

* El Selector: es el evento que debe causar el usuario para que se active la ejecución del Procesador de Comando en cuestión. Puede ser simplemente digitar un caracter o una cadena, o más generalmente, la ocurrencia de un evento más complejo causado por el usuario (p.ej. "click del ratón en una zona determinada de la pantalla").

4. "ALGORITMO" PARA DISEÑAR SISTEMAS INTERACTIVOS

Los aspectos mencionados en los numerales anteriores sobre Diagramas Estructurados de Transiciones, especificación detallada de Niveles y especificación detallada de Procesadores de Comando, pueden integrarse en un "algoritmo" de diseño de sistemas interactivos.

El punto de arranque del "algoritmo" es el símbolo distinguido, en el cual el sistema interactivo que se planea diseñar aparece como un único Procesador de Comando dependiente del Nivel 0 (el sistema operacional). Los pasos del "algoritmo de diseño" son entonces los siguientes:

Para cada procesador de comandos que no sea de tipo "batch", es decir, cuya ejecución requiera la intervención reiterada y decisiva del usuario, reemplazarlo por un nuevo Nivel (véase la 'Regla única de reescritura', numeral 2), y especificar:

i. **Tipo Abstracto asociado al Nivel**: Como se mencionó anteriormente, los Niveles son materializaciones de Tipos Abstractos de Datos (i.e. objeto + operaciones asociadas). Con frecuencia, con solo explicitar el tipo abstracto asociado al Nivel, se sugieren operaciones apropiadas para dicho tipo abstracto [BOAR 84, cap. 3].

   Por ejemplo, si en algún Nivel tuvieramos el tipo abstracto "*Conjunto de* facturas", sea cual sea el tipo de aplicación que se esté diseñando muy posiblemente se requerirán comandos como '*adicionar* factura', '*retirar* factura', '*seleccionar un subconjunto de* facturas', '*modificar* factura', ...

ii. **Variables de Estado asociadas al Nivel**: Estas variables se reconocen porque frecuentemente deberán ser consultadas y/o modificadas por los diversos Procesadores de Comando que se desprenden del Nivel. El conjunto de Variables de Estado asociadas a un Nivel está compuesto por aquella(s) que constituye(n) el objeto que se manipula en dicho Nivel más las variables que determinan la situación de manipulación.

   Por ejemplo, algunas de las variables de estado asociadas al nivel "Menú con Archivo - (edición)" que aparece en la Figura 1 podrían ser: 'texto en edición', 'fila del cursor', 'columna del cursor', 'primera línea mostrada en pantalla', 'última línea mostrada en pantalla', 'insertando o repisando', 'total de líneas', ... [BORL 86].

iii. **Valor Inicial de las Variables de Estado**: Una vez enumeradas las Variables de Estado asociadas a un Nivel, se debe establecer el valor inicial que éstas deben tomar cada vez que se acceda a dicho Nivel proveniente de un Nivel superior. Continuando con el mismo ejemplo del Nivel "menú con Archivo - (edición)" (Fig. 1), los siguientes podrían ser valores iniciales plausibles para las Variables de Estado mencionadas en el párrafo anterior:

```
'texto en edición':= contenido del archivo seleccionado
'fila del cursor':= 1
'columna del cursor':= 1
'total de líneas':= # de líneas del archivo
'insertando o repisando':= insertando
'primera línea mostrada en pantalla':= 1
'última linea mostrada en pantalla':= min   ( MAX_LINEAS_PANTALLA,
                                              # de líneas del archivo seleccionado ).
```

Es importante anotar que las labores de inicializar de las Variables de Estado y de dibujar el pantallazo inicial asociado a un Nivel son responsabilidad del procedimiento "Preparación para ..." que por construcción siempre aparece antes de cada Nivel (véase 'Regla única de reescritura', Numeral 2).

iv. **Enumeración de los Procesadores de Comando que se desprenden del Nivel:** En este punto deberán enumerarse los Procesadores de Comandos que estarán accesibles a partir del Nivel en cuestión. Muy posiblemente, aquí se requerirá ajustar especificaciones hechas en los anteriores puntos ii y iii.

v. **Especificación de la Pantalla asociada al Nivel:** Tal como se explicó anteriormente (numeral 3.1), en este punto es necesario especificar la Parte Fija, Parte Variable, y Parte Entorno que deberán aparecer en la pantalla, así como las características del interfase usuario↔ sistema para la selección de comandos.

vi. **Especificación rigurosa de los Procesadores de Comando tipo "batch" que se desprenden del Nivel:** Una vez enumerados los Procesadores de Comandos, las Variables de Estado y la disposición de la Pantalla, es posible entrar a especificar los diferentes Procesadores de Comandos en términos de PreCondiciones y PostCondiciones. En el numeral 3.2 ya se explicó cual era la forma general de estas aserciones.

5. COMPILACION DE DIAGRAMAS ESTRUCTURADOS DE TRANSICIONES

Una vez que un Sistema Interactivo ha sido diseñado, especificado y validado, se puede proceder a escribir los programas que van a implementar tal diseño. Es aquí donde se manifiesta una de las grandes ventajas de los Diagramas Estructurados de Transiciones: es posible compilarlos inmediatamente para obtener el "Esqueleto de Control" del sistema diseñado. Por **Esqueleto de Control** se entiende el programa que captura los comandos del usuario, despacha el control al Procesador de Comando o Nivel que corresponda, y posteriormente cuando recupere el control despacha el siguiente comando del usuario.

En los siguientes numerales se presentan varios esquemas canónicos de compilación de niveles para diversos tipos de interfase usuario↔ sistema.

Obviamente existen muchos más esquemas de interfase usuario↔ sistema que no son cubiertos con los casos que se presentan a continuación. Sin embargo, se ha visto que prácticamente todos los esquemas responden a los mismos principios.

5.1 *Compilación Canónica de Niveles con retorno único y Selección Directa*

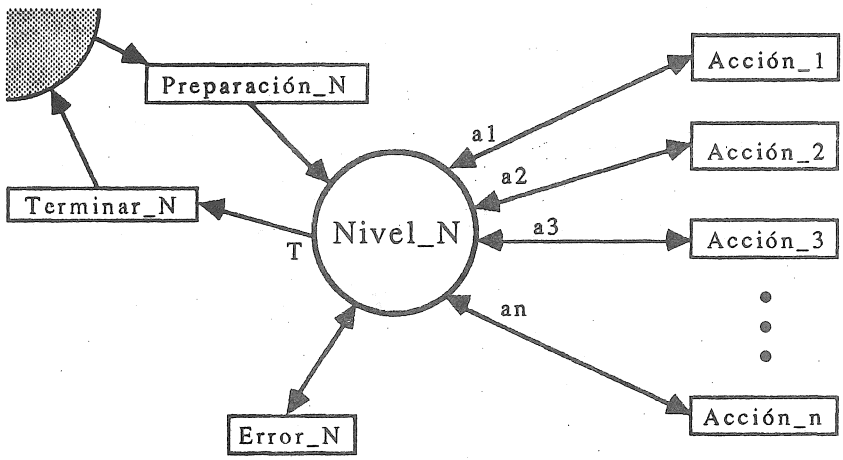


Figura Nº 2: Nivel con retorno único

Supongamos que al Nivel_N del anterior subdiagrama se le asocia un mecanismo muy simple de Selección Directa: cada comando está identificado por una letra, y el usuario selecciona el comando deseado tecleando la letra respectiva. En estas condiciones, el subdiagrama de la Figura Nº 2 se compilaría así:

```
PROCEDURE Nivel_N;              /* esqueleto de control de un nivel con */
VAR                             /* retorno único y selección directa    */
   c  :  char;
BEGIN
Preparación_N;
read(c);
while (c <> 'T') do begin
   case c of
      a1 : Acción_1;
      a2 : Acción_2;
      ..   ...

      an : Acción_n;
      ELSE Error_N;
   end;
   read(c);
end;
Salida_N;
END;
```

5.2 *Compilación Canónica de Niveles con retorno múltiple y Selección Directa*

Consideremos ahora el siguiente SubDiagrama:

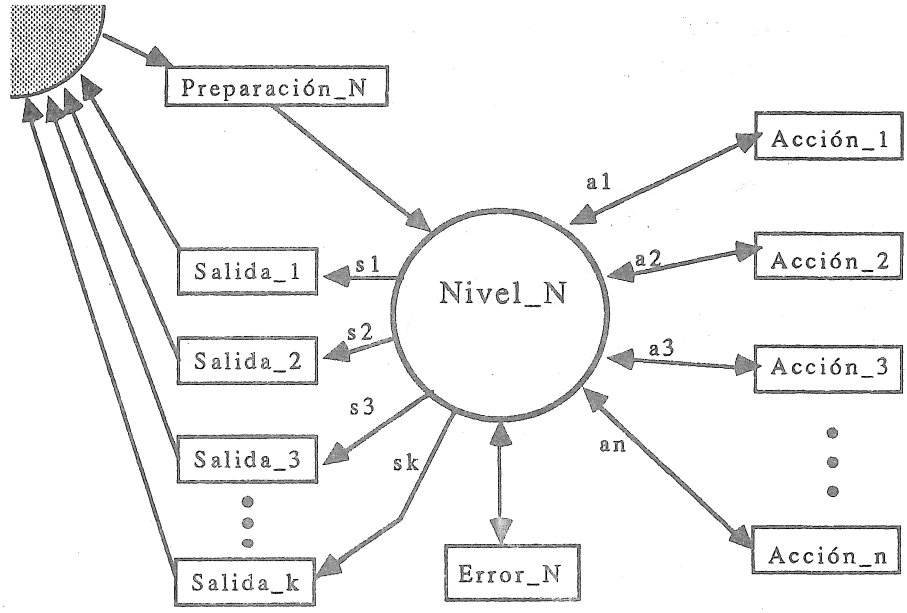


Figura Nº 3: Nivel con retorno múltiple.

Suponiendo de nuevo el mismo esquema de selección directa descrito en el numeral 5.1, el subdiagrama de la Figura Nº 3 se compilaría así:

```
PROCEDURE Nivel_N;                  /* Esqueleto de Control de un Nivel con */
VAR                                 /* retorno múltiple y selección directa */
  c        : char    ;
  fin_N    : boolean;
BEGIN
Preparación_N;
fin_N:= FALSE;
repeat
   read(c);
   case c of
      a1:  Acción_1;
      ..   ...
      an:  Acción_n;
      s1:  begin  Salida_1; fin_N:= TRUE  end;
      ..   ...
      sk:  begin  Salida_k; fin_N:= TRUE  end;
      ELSE Error_N;
   end;
until fin_N;
END;
```

5.3 *Compilación Canónica de Niveles con Retorno Múltiple, Preselección y Selección Directa*

Consideremos de nuevo el subdiagrama de la Figura Nº 3, pero esta vez con un mecanismo un poco más complejo para selección de comandos: Selección Directa más Preselección-Confirmación. En otras palabras, supongamos que en cada interacción el usuario puede:

- cambiar la preselección con teclas de dirección (⬆ ⬇ ⬅ ➡), ó
- activar la ejecución de la acción preseleccionada oprimiendo alguna tecla de confirmación **OK**, ó
- activar la ejecución de cualquier acción oprimiendo su selector directo.

Es deseable que la acción que se encuentre preseleccionada aparezca resaltada en la pantalla. Cuando el usuario cambie de preselección con las teclas de dirección, el resaltado debe desplazarse a la nueva preselección. Adicionalmente, cuando el usuario seleccione una acción oprimiendo su selector directo, el resaltado debe quitarse de donde se encontraba, colocarse sobre la acción que acaba de ser seleccionada directamente, esperar allí algunas décimas de segundo, y posteriormente activar la acción respectiva.

Siguiendo las anteriores descripciones, el subdiagrama de la Figura Nº 3 se compilaría en el procedimiento mostrado en la página siguiente.

---*---

Cabe anotar que los ejemplos de esqueletos de control que se han mostrado son "compilaciones canónicas", es decir, salvo pequeños detalles de sistema operacional, tipo de pantalla o compilador, realmente compilan el esqueleto de control del Nivel en cuestión. Al ciclo principal de estas compilaciones canónicas se le suele llamar *"Despachador de Comandos"*.

Sinembargo, este aspecto de la compilación canónica de Niveles admite un gran número de variaciones. El factor que genera estas variaciones es el tipo de interfase usuario↔ sistema para selección de comandos. De todas maneras, para una buena cantidad de esquemas se ha visto que se puede construir una variante apropiada a partir de alguna de las compilaciones canónicas que se han mostrado anteriormente.

```
PROCEDURE Nivel_N;                  /* Esqueleto de Control de un Nivel con */
VAR                                 /* retorno múltiple,selección directa y */
   c        : char;                 /*      preselección + confirmación     */
   fin_N    : boolean;
   presel   : (a1,a2,...,an,s1,s2,...sk);
BEGIN
Preparación_N;
fin_N:= FALSE;
presel:= a1;
resalte(presel);
repeat
   read(c);
   case c of
      →:   SucesorHorizontal(preselec);
      ↓:   SucesorVertical(preselec);
      ←:   PredecesorHorizontal(preselec);
      ↑:   PredecesorVertical(preselec);
      OK:  case preselec of
              a1:  Acción_1;
              ..   ...
              an:  Acción_n;
              s1:  begin  Salida_1; fin_N:= TRUE  end;
              ..   ...
              sk:  begin  Salida_k; fin_N:= TRUE  end;
           end;
      a1:  begin  arregle(preselec,a1); Acción_1  end;
      ..:  ..;
      an:  begin  arregle(preselec,an); Acción_n  end;
      s1:  begin  arregle(preselec,s1); Salida_1; fin_N:= TRUE  end;
      ..:  ..;
      sk:  begin  arregle(preselec,sk); Salida_k; fin_N:= TRUE  end;
      ELSE Error_N;
   end;
until fin_N;
END;
```

donde:

```
PROCEDURE SucesorHorizontal (VAR  ActualPreselec  :  ...);
BEGIN
normalice(ActualPreselec);
ActualPreselec:= ....{fórmula o tabla para determinar el sucesor };
resalte(ActualPreselec);
END;

{Los procedimientos SucesorVertical, PredecesorHorizontal y
 PredecesorVertical tienen exactamente la misma estructura que el
 anterior. Varían únicamente en la forma de determinación del nuevo valor
 de 'ActualPreselec' (asignación de la 2ª línea)
}
```

```
PROCEDURE arregle (VAR  ActualPreselec  :  ...;
                        NuevaPreselec   :  ... );
BEGIN
normalice(ActualPreselec);
resalte(NuevaPreselec);
ActualPreselec:= NuevaPreselec;
delay( )
END;
```

6. PROTOTIPIFICACION DE SISTEMAS INTERACTIVOS

Uno de los nuevos postulados para el desarrollo de sistemas de software de mediana complejidad es la incorporación directa del usuario en las fases de diseño y especificación. Esto se trata de lograr a través de la elaboración de prototipos rápidos en los que se van explicitando y consensando las necesidades reales de los usuarios. Dichos prototipos deberán evolucionar en sucesivas versiones hasta culminar en el sistema definitivo [BOAR 84][AGRE 86].

Antes de plantear en términos generales la metodología de prototipificación, considérese inicialmente el siguiente Diagrama Estructurado de Transiciones:

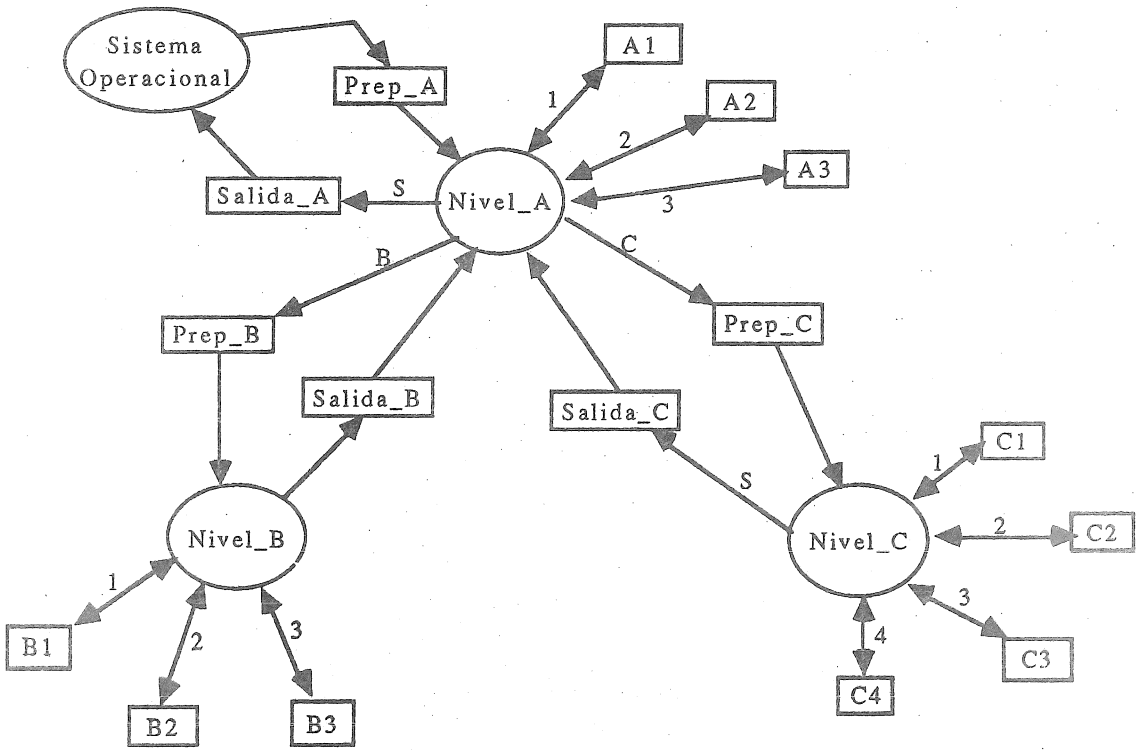


Figura Nº 4: Un Diagrama de Transiciones de profundidad dos

Observando con atención se verá que el siguiente programa resuelve todo el problema de captura y despacho de control asociado al diagrama de la Figura Nº 4 (¡¡ independientemente de cual sea la semántica de los diversos Procesadores de Comando !!).

```
PROGRAM Figura_4 (input, output);    { Esqueleto del Sistema Interactivo }
                                     {     descrito en la Figura Nº 4    }
VAR
   c  :  char;

{-------------------------------------}

PROCEDURE Prep_A;  BEGIN .... END;
PROCEDURE A1;  BEGIN .... END;
PROCEDURE A2;  BEGIN .... END;
PROCEDURE A3;  BEGIN .... END;
PROCEDURE ErrorNivel_A;  BEGIN .... END;
PROCEDURE Salida_A;  BEGIN .... END;

PROCEDURE Prep_B;  BEGIN .... END;
PROCEDURE B1;  BEGIN .... END;
PROCEDURE B2;  BEGIN .... END;
PROCEDURE B3;  BEGIN .... END;
PROCEDURE ErrorNivel_B;  BEGIN .... END;
PROCEDURE Salida_B;  BEGIN .... END;

PROCEDURE Prep_C;  BEGIN .... END;
PROCEDURE C1;  BEGIN .... END;
PROCEDURE C2;  BEGIN .... END;
PROCEDURE C3;  BEGIN .... END;
PROCEDURE C4;  BEGIN .... END;
PROCEDURE ErrorNivel_C;  BEGIN .... END;
PROCEDURE Salida_C;  BEGIN .... END;

{-------------------------------------}

PROCEDURE Nivel_B;
BEGIN
Prep_B;
read(c);
while c <> 'S' do begin
   case c of
      '1': B1;
      '2': B2;
      '3': B3;
      else ErrorNivel_B
   end;
   read(c)
end;
Salida_B
END;

{-------------------------------------}
```

```
{--------------------------------------}

PROCEDURE Nivel_C;
BEGIN
Prep_C;
read(c);
while c <> 'S' do begin
   case c of
      '1': C1;
      '2': C2;
      '3': C3;
      '4': C4;
      else ErrorNivel_C
   end;
   read(c)
end;
Salida_C
END;

{**************************************}
{********  PROGRAMA PRINCIPAL *********}
{**************************************}
BEGIN
Prep_A;
read(c);
while c <> 'S' do begin
   case c of
      '1': A1;
      '2': A2;
      '3': A3;
      'B': Nivel_B;
      'C': Nivel_C;
      else ErrorNivel_A
   end;
   read(c)
end;
Salida_A
END.
```

En términos generales, el desarrollo de un prototipo consta de tres etapas: primero escribir el **Esqueleto del Sistema**, luego definir y maquillar la interfase usuario↔sistema, y posteriormente ir implementando los diversos servicios.

Se toma entonces como punto de partida un Diagrama Estructurado de Transiciones suficientemente especificado (véase numeral 3), y se procede a compilar el Nivel superior (i.e. aquel único Nivel que desciende directamente del Sistema Operacional). El código obtenido se constituye en el programa principal. Posteriormente se procede a compilar cada uno de los otros Niveles del Diagrama. Se irá obteniendo así una serie de procedimientos despachadores de control que son invocados o bien por el programa principal o bien por otros procedimientos despachadores de control.

Obviamente, el encapsulamiento del programa principal más los procedimientos despachadores de control obtenidos al compilar todos los Niveles no constituye aún un programa fuente completo. Hace falta resolver todas las referencias a los procedimientos Procesadores de Comando que figuran en los códigos obtenidos al compilar los Niveles. Para ésto, se debe proceder a crear un procedimiento, inicialmente vacío, por cada Procesador de Comando que aparezca en el Diagrama de Transiciones. El programa obtenido al encapsular el programa principal, los procedimientos que compilan los Niveles, y los procedimientos vacios correspondientes a los Procesadores de Comandos constituye el Esqueleto del **Sistema Interactivo.**

El programa Esqueleto obtenido es compilable y ejecutable, y permite que el control visite cualquier nivel o procesador de comandos. Sin embargo, aún no permite que el usuario valide la información desplegada en los diferentes niveles, ni perciba y evalue la apariencia de la interfase. Para esto, como mínimo sería necesario rellenar los procesadores de comandos que conectan dos Niveles diferentes con el código necesario para desplegar el pantallazo del Nivel de llegada (en el ejemplo de la Figura Nº 4, estos serían los procedimientos 'Prep_A', 'Prep_B', 'Prep_C', 'Salida_A', 'Salida_B' y 'Salida_C'). De esta manera, se obtendría un primer prototipo 'cascarón', que permite visitar cualquier Nivel o Procesador de Comandos, y facilita que el usuario se forme una idea global del tipo y forma de la información que se desplegará en pantalla en cada Nivel.

En esta primera fase de prototipificación es importante postergar al máximo la producción de código atado a un tipo específico de interfase o a una distribución particular en la pantalla de la información presentada en un Nivel. De hecho, son justamente los aspectos de interfase y presentación de la información los que suscitan el mayor número de sugerencias o críticas de los usuarios durante ltodo el desarrollo del sistema.

Para obviar este problema hay un método muy sencillo que nos ha dado excelentes resultados: con un editor de texto normal se crean una a una configuraciones plausibles de los diversos pantallazos, los cuales son guardados en archivos separados. Posteriormente se coloca en cada uno de los procedimientos que conectan Niveles una invocación del estilo:

```
MuestrePantallazo ( nombre del archivo que contiene el pantallazo asociado al Nivel de llegada ) ;
```

donde 'MuestrePantallazo' simplemente copia en pantalla el contenido del archivo especificado. Adicionalmente, se pueden establecer algunos caracteres de control para indicar cambios de color o de atributos de video, los cuales serían interpretados por 'MuestrePantallazo' al momento de estar haciendo el eco a la

pantalla. De esta manera, bastaría con editar y modificar el archivo para reflejar los cambios de forma que sugiera el usuario. Posteriormente, partir de este primer prototipo "cascarón" pueden irse desarrollando en forma evolutiva versiones cada vez más sofisticadas del prototipo, hasta converger al sistema definitivo [BOAR 84][RIVE 87].

CONCLUSIONES

En este trabajo se ha presentado un lenguaje para describir globalmente Sistemas Interactivos. Posteriormente se han descrito cada uno de los pasos que se deben recorrer para diseñar y especificar dichos sistemas. En seguida se mostró como se pueden procesar tales diseños en forma cuasi-automática para obtener el Esqueleto del Sistema Interactivo. Finalmente, se vió como se puede construir muy rápidamente un prototipo "cascarón" que permita al usuario tener un primer contacto "real" con el sistema en desarrollo, siendo evidente que posteriores refinamientos de dicho prototipo pueden incorporarse ordenadamente en el Esqueleto del Sistema Interactivo.

La piedra angular de la metodología son los Diagramas Estructurados de Transiciones. Estos Diagramas, definidos rigurosamente por una gramática formal, establecen claramente el espacio de posibilidades en el que mueve el diseñador. Es importante anotar que otros trabajos están explorando extensiones "non sanctas" a dichos diagramas, algunas de las cuales son de gran utilidad para el tratamiento y recuperación de errores [LOPE 87b]. Está aún pendiente la especificación y el esquema de compilación cuasi-automática de situaciones que frecuentemente ocurren en la práctica, como por ejemplo la anulación o interrupción por parte del usuario de la ejecución normal de los procesadores de comando.

Si bien todos los ejemplos de código se hicieron en Pascal, cabe anotar que las compilaciones canónicas admiten codificaciones muy elegantes en el lenguaje C. De todas maneras, la similitud de los diversos esquemas canónicos de compilación de Niveles sugiere una extensión hacia manejadores generalizados de Niveles. De hecho, en 'C' pueden escribirse en forma condensada, elegante y eficiente manejadores generalizados que pueden procesar diversos Niveles vía una adecuada parametrización [KALY 85].

Actualmente el grupo "ISAC: Ingeniería de Software Apoyada por Computador" está trabajando en la construcción de un ambiente de soporte a la metodología aquí descrita. Tales soportes incluyen un editor/compilador de Diagramas de Transiciones, un editor/compilador de pantallazos [RIVE 87], así como diversas librerías de manejo de interfases.

BIBLIOGRAFIA

[AGRE 86] Agresti, W. W.
"Tutorial: New Paradigms for Software Developpment"
IEEE Computer Society Press 1987
Código IEEE: EH0245-1; Código ISBN: 0-8186-0707-6

[BARN 86] Barnard, J.; Metz, R.; Price, A.
"A Recommended Practice for Describing Software Designs: IEEE Standards Project 1016"
IEEE Transactions on Software Engineering, Vol. SE-12, Nº 12
Febrero 1986

[BERG 81] Bergland, G.
"A Guided Tour for Program Design Methodologies"
IEEE Computer, Vol. 14, Nº 10, Octubre 1981

[BOAR 84] Boar, B. H.
"Application Prototyping: a requirement definition strategy for the 80's"
John Wiley & Sons, 1984

[BOHO 86] Bohórquez, J. A.
"Cálculo de Programas: una Metodología para el Diseño de Programas de Computador"
- UniAndes-CIFI, Memo de Investigación Nº 6, 1986
- XII Conferencia Latinoamericana de Informática CLEI, Montevideo, Noviembre 1986

[BORL 86] Borland International Inc.
"Turbo Editor Toolbox - Owner's Handbook" ©1985

[CARD 86] Cardoso, R.
"Diseño e Implementación de Tipos Abstractos de Datos: una Metodología basada en Funciones Recursivas"
- UniAndes-CIFI, Memo de Investigación Nº 9, 1986
- XII Conferencia Latinoamericana de Informática CLEI Montevideo, Noviembre 1986

[CARD 87] Cardoso, R.
"Prácticas en Tipos Abstractos de Datos"
- UniAndes-CIFI, Memo de Investigación Nº 17, 1987
- XII Conferencia Latinoamericana de Informática CLEI
Montevideo, Noviembre 1986

[GEHA 86] Gehani, N. (Ed.)
"Software Specification Techniques"
Addisson Wesley, 1986

[GRIE 81] Gries, D.
"The Science of Programming"
Springer Verlag, 1981

[KALY 85] Kalyan, D.
"Modular Programming in C: an approach and an example"
Sigplan Notices, Vol. 20, Nº 3, pp. 9 - 15, Marzo 1985

[LOPE 87a] López, R.; Toro, V. M.
"turbo-KAREL (versión 1.5): Libro del Proyecto"
Informe final del Contrato FC-009
Fonade-Secretaría de Informática de la Presidencia de la República-
Universidad de los Andes. Mayo de 1987.

[LOPE 87b] López, R.
"Saltos no Locales: una extensión a Turbo Pascal"
XIII Conferencia Latinoamericana de Informática
Bogotá, noviembre de 1987

[MYER 78] Myers, G.
"Composite Structured Design"
Van Nostrand, 1978

[PRES 87] Pressman, R. S.
"Software Engineering: a Practitioner's Approach" (2ª edición)
McGraw Hill, 1987

[RIVE 87] Rivera, G.; Reyes, I. H.
"Cartelera Dinámica: diseño e implementación"
XIII Conferencia Latinoamericana de Informática
Bogotá, noviembre de 1987

[YOUR 76] Yourdon, E.; Constantine, L.
"Structured Design: Fundamentals of a Discipline of Computer Programs and System Design"
Prentice-Hall (Yourdon Press), 1976

[YOUR 86] Yourdon, E.
"What ever happened to Structured Analysis ?"
Datamation, junio 1986, págs. 133-138

[WASS 81] Wasserman, A. I.
"User Software Engineering and the Design of Interactive Systems"
5th International Conference Software Engineering (IEEE)
San Diego CA, marzo 1981.

---*---

CARTELERA DINAMICA:
Diseño e Implementación

J. GERMAN RIVERA A. IVAN H. REYES G.

Estudiantes de Ingeniería de Sistemas
Universidad de los Andes
Apartado Aéreo 4976
Bogotá-Colombia

La CARTELERA DINAMICA es una herramienta diseñada para la aplicación del computador en la difusión de información al público en eventos donde fuese necesario. Incluye un versátil editor y un compilador de pantallazos, los cuales constituyen una herramienta muy útil para el diseño de interfases de sistemas interactivos alfanuméricos.

INTRODUCCION.

La CARTELERA DINAMICA es una herramienta diseñada para la aplicación del computador en la difusión de información al público en eventos tales como congresos y exposiciones, ferias, aeropuertos, etc. Fue desarrollada en el departamento de Ingenieria de Sistemas y computación de la Universidad de los Andes como proyecto del curso "Ingenieria de Software" durante el primer semestre del año en curso, y posteriormente afinada para su utilización en la XIII conferencia Latinoamericana de Informática.

La CARTELERA DINAMICA maneja tres tipos de objetos : PANTALLAZOS, que son imágenes alfanuméricas que utilizan los diversos atributos de video; SECUENCIAS, que son sucesiones de PANTALLAZOS a intervalos de tiempo preestablecidos, y PROGRAMAS que contienen un determinado número de SECUENCIAS que aparecen también durante un número de veces preestablecido.

El diseño inicial se hizo para el manejo de estos objetos, sin embargo durante su desarrollo se encontraron otras posibles aplicaciones o derivaciones. Se desarrolló entonces un Editor de "Pantallazos" y se escribió una herramienta auxiliar que permite codificar automáticamente los pantallazos. Estas dos herramientas aumentaron considerablemente la versatilidad de la CARTELERA DINAMICA dejando además abierta la posibilidad de ampliar las aplicaciones en el área de diseño de Software interactivo.

Por último, la CARTELERA DINAMICA esta diseñada para ser usada en computadores PC-Texas Instruments está en curso la version PC-IBM.

1. DESCRIPCION DEL SISTEMA.

1.1 Qué es la CARTELERA DINAMICA (C.D) ?

Es una herramienta de Software cuya aplicación principal está en la difusión de información al público en eventos tales como congresos y exposiciones, y con fines publicitarios en ferias y aeropuertos por ejemplo. Para lograr este propósito es necesario que al computador donde se instale la CARTELERA DINAMICA se conecten sendos monitores o pantallas de televisión a través de los cuales se pueda transmitir la misma información que se observa en la pantalla del computador.

1.2 Quienes la desarrollaron ?

En el desarrollo inicial de la C.D. colaboraron los estudiantes Doris Pinzón, Nancy León y Humberto Ruíz. Posteriormente los autores, bajo la dirección del profesor Víctor Toro, la afinaron para su utilización en la XIII Conferencia Latinoamericana de Informática.

1.3. Objetos que se manipulan.

La C.D. manipula tres tipos de objetos :

PANTALLAZOS : imagen alfanumérica que aparece inmóvil en una pantalla de video. Utiliza todos los recursos de atributos y color.

SECUENCIAS : sucesión de PANTALLAZOS que se van desplegando en la pantalla durante un intervalo de tiempo preestablecido. A cada uno de los pantallazos se le asigna un modo de aparición.

PROGRAMAS : sucesión de SECUENCIAS que "desfilan" por la pantalla un cierto número de veces cada una.

1.4. Facilidades Generales que ofrece.

En el nivel de Pantallazos se permiten efectuar las siguientes operaciones (sobre Pantallazos) :

a) Seleccionar . Operación que como su nombre lo indica permite escoger el nombre del pantallazo sobre el que se efectuarán otras operaciones. Puede tratarse de uno ya existente o de uno que hasta ahora se va a "crear". Para el primer caso el sistema tan pronto como entra a este nivel muestra en la parte superior derecha de la pantalla una lista (directorio) de los pantallazos que existen en el diskette que se encuentra en el drive "activo".

b) Editar. Operación que invoca al Editor de Pantallazos, el cual se encarga del manejo de los atributos de video, color y caracteres especiales para lograr en el diseñno de los Panta-

llazos máxima eficacia. Cabe anotar que en la versión actual no es posible hacer gráficas.

c) Guardar. Permite guardar en diskette el Pantallazo, donde queda incluido como cualquier otro archivo que para el caso se identifica por la extensión "pan".

d) Borrar. Permite borrar del diskette el pantallazo seleccionado, si ya existe.

e) Renombrar. Si se desea cambiar el nombre del pantallazo seleccionado se usa esta opción.

Las opciones anteriores son análogas en los niveles de Secuencias y Programas.

El Editor de Secuencias permite crear una lista que contiene en cada componente el nombre del pantallazo, el tiempo en segundos que se desea mantener en la pantalla y el modo como debe aparecer en la misma: instantáneo, en espiral hacia adentro, en espiral hacia afuera, aparición "móvil" por cualquiera de los costados de la pantalla o en forma aleatoria.

Por su parte el Editor de Programas permite especificar en cada elemento de la lista el nombre de una secuencia y el número de veces que se desea repetir.

En ambos Editores se ofrece la facilidad de tener un directorio de los respectivos objetos disponibles. Así, el editor de Secuencias tiene un directorio de Pantallazos, y el de Programas uno de Secuencias.

Las operaciones adicionales y propias de estos dos niveles son:

f) Verificar . Que permite determinar si una Secuencia o un Programa tiene errores sintácticos o inconsistencias tales como incluir un pantallazo o secuencia inexistentes, asociar un tipo de aparición no definido, etc.

g) Ejecutar . Que en cada caso muestra como se verá "en acción" una Secuencia o un Programa.

Adicionalmente e independiente del objeto seleccionado se dispone de las siguientes opciones :

h) Directorio . Que permite ver la lista de objetos disponibles en el diskette "activo".

i) Drive Activo . Opción particularmente útil cuando se tienen diskettes con objetos disponibles en ambos drives y se desea trabajar sobre alguno de los dos.

j) ESC . Para abandonar el nivel donde se encuentra y regresar al nivel que controla el sistema en general.

2. ESTRUCTURA DEL SISTEMA.

La C.D. está compuesta de 4 partes principales:

El Distribuidor del Control . Contiene el Menú con las opciones principales: cada uno de los Manejadores de los objetos que se manipulan.

La forma totalmente interactiva de presentación del Menú permite al usuario entender muy fácilmente la forma como debe accesar cada una de las opciones propuestas. Además, se cuenta con un sistema novedoso de "navegación" por el menú que llama la atención y despierta el interés del usuario desde el primer momento.

Los demás componentes son los Manejadores asociados a cada uno de los objetos ya descritos : Manejador de Pantallazos, Manejador de Secuencias y Manejador de Programas . Los elementos que los constituyen son los que permiten el manejo de las facilidades descritas en el punto anterior con particular énfasis en los editores, verificadores y ejecutadores cuyas funciones ya se mencionaron.

Sin embargo, cabe anotar que estos manejadores estan provistos de un sistema que le permite al usuario, sin mayor esfuerzo, realizar todas las operaciones necesarias hasta obtener el "objeto terminado" en cuestión. Explicaremos el caso del manejador de Secuencias que funciona exactamente igual al de Programas. El de Pantallazos es análogo pero con dos operaciones menos.

El sistema está hecho de manera que no es posible en cualquier momento escoger cualquiera de las opciones que ofrece el Manejador. Así, al entrar inicialmente al Manejador se observa en la parte media izquierda de la pantalla la lista de opciones, de las cuales sólo es posible en principio escoger la opción Seleccionar.

Es decir, las demás opciones se encuentran en ese momento "inhabilitadas", ya que para el caso no tiene sentido tratar de realizar cualquiera de estas operaciones sobre un objeto que aún no se ha Seleccionado. Una vez hecho esto, se "habilitan" las opciones Editar, Guardar y Renombrar en el caso de que se trate de un objeto (Secuencia en este caso) nuevo. De lo contrario se "habilitan" todas.

Si es el primer caso, una vez creada y guardada la Secuencia seleccionada se "habilitan" las demás opciones. Esta forma de manejar las opciones le ayuda a entender al usuario qué puede y que no puede hacer en un momento cualquiera. Además, se cuenta con un sistema de guía de lo que hemos denominado la "Preselección".

La opción que está indicada por las "Flechas" es la Preselección. Cada vez que se escoge una opción, al salir de ella la Preselección se ha variado indicándole al usuario que es lo que "deberia" hacer a continuación. Así por ejemplo, después de Seleccionar, la

Preselección indica "Editar", y después de esta la Preselección indicará "Guardar".

El usuario es libre de aceptar la "sugerencia" o no, pero en cualquier caso este sistema proporciona una forma muy flexible de interacción .

3. CARACTERISTICAS DE LA IMPLEMENTACION.

La única implementación de la C.D. que está en total funcionamiento es para computadores T.I "Professional Computer".Se hizo en lenguaje C-Lattice y entre las características especiales que posee podemos citar las siguientes :

- Aprovecha al máximo la arquitectura del sistema de video Alfanumérico.Esto permite manejar velózmente la pantalla (los Pantallazos aparecen casi instantáneamente). El diseño de la arquitectura de video es relativamente sencilla: la pantalla virtual (memoria asociada a la pantalla) contiene el código de los caracteres. Los atributos con que aparece el caracter en pantalla están en un Latch (registro). De acuerdo con esto, cada vez que se desee cambiar los atributos de un caractér o de una cadena de caractéres se debe actualizar el contenido del Latch con el valor apropiado.

- La parte no portable a otros computadores MS-DOS está bien aislada del resto y por lo tanto, es fácil hacer las modificaciones necesarias siempre que se cuente con la información propia del computador para poder implantarse las rutinas básicas de video.

- Todos los pantallazos de la interfase con el usuario fueron diseñados con el propio Editor de Pantallazos de la C.D. Este hecho es a nuestro parecer realmente novedoso, porque muestra cómo un software ayuda a construirse a si mismo.

- La C.D. tiene una gran solidez ante los errores del usuario, como dejar un drive con la puerta abierta, pulsar la tecla BREAK por accidente, etc. Esto la hace muy confiable, colocandola al mismo nivel de los paquetes comerciales.

4. PERSPECTIVAS DE DESARROLLO.

Aunque la finalidad con la que se hizo la CARTELERA DINAMICA era que sirviera como medio de información al público,durante su desarrollo se encontró que el Editor de Pantallazos podia ser lo suficientemente potente como para usarlo como una herramienta independiente en el diseño de pantallazos para interfases en sistemas interactivos en general, tal y como se hizo con la misma C.D.

Este motivo la creación de otra herramienta que permite que los pantallazos creados puedan ser usados desde un programa sin necesidad que el programador tenga que codificarlos como parte del programa. Es decir, una herramienta que genere automáticamente el código fuente que escribe un pantallazo.

Con la "compilación" automática de pantallazos se puede ir mucho más lejos de lo que aquí se logró. Una posibilidad que se puede explotar es la de sofisticar mas el editor de Pantallazos y el generador de código fuente, de tal forma que se pudeda especificar en que partes de la pantalla se deben refrescar variables de estado y después generar el código fuente de las rutinas encargadas de hacerlo. Esto proporcionaría una base para un sistema que permitiera la creación de ambientes para el desarrollo casi automático de sistemas interactivos, obteniendo así una herramienta de incalculable valor dentro del ámbito del desarrollo de Software.

5. BIBLIOGRAFIA.

LATICCE C COMPILER FOR MS-DOS
Version 3. Programmer's Reference Manual.

TECHNICAL REFERENCE MANUAL.
Profesional Computer - Texas Instruments.

APUNTES DE CLASE.
Ingeniería de Software Universidad de los Andes - 1987

Arquitectura de una Interfase Usuario: Un Modelo General

Jorge A. Villalobos S.

Grupo de Investigación CGI/DAC
Departamento de Sistemas y Computación
Universidad de los Andes
Apartado Aéreo 4976 - Bogotá, COLOMBIA

Resumen: En este artículo se presenta un modelo general de una Interfase Usuario. El objetivo es definir un esquema da tal manera que cualquier tipo de interacción entre una aplicación y un usuario sea facilmente expresable en términos del modelo. Esto implica que puede manejar cualquier tipo de dispositivos, y cualquier metodología de interacción (menus, comandos, etc.), lo mismo que eventos sincrónicos y asincrónicos, ayudas a todos los niveles, recuperación en caso de error, y todas aquella otras caracteristicas deseables en una interfase usuario.

El presente escrito constituye la continuación de un trabajo desarrollado por el autor en el Institut National Polytechnique de Grenoble [VIL86].

Palabras Clave: Arquitectura de Interfases Usuario, Diseño de Interfases Usuario, Interacción hombre-máquina, Ingeniería de Software.

1. Introducción.

La interfase usuario es aquella parte de una aplicación que permite la interaccion entre el usuario y el resto del programa. El problema que existe actualmente es que debido a la masificación de la informática y a las grandes posibilidades hardware de interacción con el usuario, el diseño y la programación de la interfase están alcanzando costos considerables, llegando en algunos casos a constituir el 50% del codigo total de un programa.

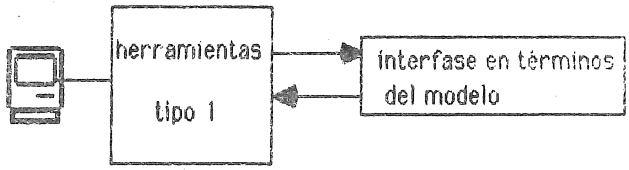
En el sistema tradicional de desarrollo de programas interactivos, el programador recibe una breve descripción de la interfase. A partir de ella, y utilizando lenguajes de programación que no cuentan a menudo con herramientas poderosas para este tipo de trabajo, debe implementar toda la interfase hasta niveles muy cercanos al nivel físico. Cada uno utiliza entonces su propio estilo y su propia interpretación de la especificación. Además, por la misma concepción de la aplicación, la interfase se encuentra repartida a lo largo de todo el programa haciendo imposible su adecuado mantenimiento y modificación. Esto conlleva en general muchos problemas sobre el producto final: dificultad en el mantenimiento de la interfase, imposibilidad de evaluar una especificación sin hacer su respectivo desarrollo, alto riesgo de error en la implementación, etc., además de las innumerables dificultades que existen para hacer una buena especificación de la interacción usuario-aplicación

El primer paso que se ha dado para superar estos problemas ha sido el de utilizar métodos formales de especificación (gramáticas, BNF, autómatas), pero aun contando con una buena definición de la interfase el costo de su implementación sigue siendo alto y los problemas de mantenimiento quedan sin solución. Lo ideal en ese caso es poder ejecutar directamente la especificación o al menos poder generar automaticamente, a partir de ella, el código necesario para su ejecución.

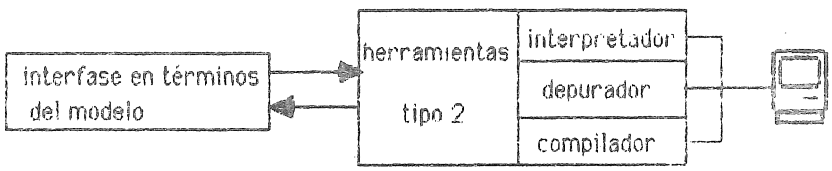
En este artículo se pretende mostrar un modelo general de interfases usuario, basado en el modelo de niveles propuesto por Olsen [OLS83, OLS84] y Green [GRES5], y seleccionar un formalismo para poder representar cada nivel. También se incluye la definición de un esquema de comunicación entre ellos

2. Para qué el modelo?

El objetivo final del proyecto que se está desarrollando, es la creación de un ambiente completo para la especificación y generación de interfases usuario. Para esto se ha decidido que la mejor manera de enfrentar el problema de diseñar las herramientas del ambiente de programación es crear como primera medida un modelo general de una interfase usuario y luego crear dos tipos de herramientas. las primeras que le permitan al programador (el usuario del sistema) expresar la interfase que desea desarrollar en términos del modelo. Si el modelo está bien construido, al igual que las herramientas del ambiente, el programador debe ser capaz de expresar fácilmente la interfase de cualquier sistema sin importar su tipo.



El segundo tipo de herramientas trabajan sobre cualquier interfase que se encuentre expresada en términos del modelo; le permiten al programador experimentar sobre ella (simular su ejecución, depurarla) y generar código en algún lenguaje de programación de alto nivel, que la ejecute.



El modelo debe entonces manejar de manera natural cualquier tipo de interacción entre el usuario y la aplicación, lo mismo que sistemas de ayuda dependiente del contexto (a varios niveles), recuperación en caso de error, cancelacion de acciones etc

Resumiendo se puede decir que el modelo se hizo para definir las caracteristicas y estructura del ambiente de programación de interfases, y en general para guiar la construcción de herramientas para la ayuda al diseño y creación de interfases usuario.

3. Características de una buena herramienta de diseño.

Las características que se consideran indispensables para cualquier herramienta de diseño de interfases (y en general para el diseño de software), y que nos deben por lo tanto guiar en la creación del modelo son las siguientes cuatro:

- El diseño y la especificación deben ser más fáciles de entender, crear y mantener que el programa que los ejecuta. La razon es que si es mas dificil diseñar la interfase que programarla directamente, no vale la pena utilizar las herramientas propuestas.

- Las herramientas de diseño deben ser suficientemente poderosas para poder especificar la interfase de cualquier sistema. Esto quiere decir que no se puede orientar hacia ningún tipo especifico de interacción (existen herramientas que solo trabajan sobre menus, por ejemplo), ni hacia ninguna clase particular de hardware

- Las herramientas deben permitir la ejecución directa de la especificación de la interfase o al menos la generación del código correspondiente. Sin esto pierde un poco de sentido el gastar tanto tiempo en la tarea de crear una buena especificación, puesto que los problemas se van a presentar luego en las fases de implementación y mantenimiento.

- El modelo en el que se basan las herramientas debe estar fuertemente ligado al modelo mental que el programador asocia a la interfase, o más exactamente al proceso de su diseño. Esto implica que se descartan de plano las gramáticas y las BNF propuestas en algunos sistemas, dada su dificultad de utilización para especificar una interfase.

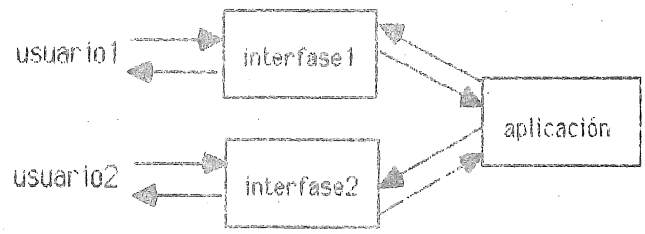
Teniendo en cuenta estas características y otras consideraciones como portabilidad, modularidad, etc. que exige la ingeniería de software, se propone el siguiente modelo para una interfase usuario.

4. Estructura general del modelo.

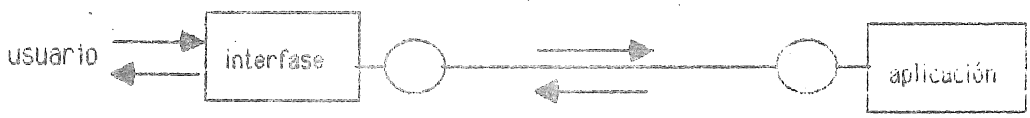
De acuerdo con la definición dada, la interfase es la parte del sistema encargada de la interacción entre la aplicación y el usuario.

Para poder garantizar independencia y modularidad de la interfase, vamos a exigir una separación clara entre ésta y la aplicación, encontrándose relacionadas únicamente mediante un esquema formal de comunicación.

Esto quiere decir que una misma aplicación puede ser manejada por dos interfases distintas (dependiendo posiblemente del tipo de usuario) sin necesidad de ser modificada.



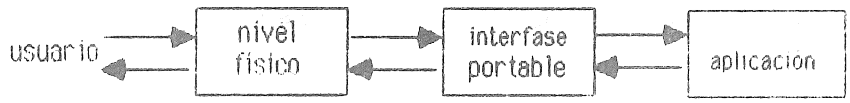
llegando incluso en un sistema distribuido a permitir que la interfase se encuentre en el nodo del usuario y la aplicación en un nodo diferente de la red,



sin necesidad de variar ni la aplicación ni la interfase, sino únicamente adoptando un nuevo esquema de comunicación entre los dos.

Además de éso es necesario, para poder garantizar independencia del hardware (portabilidad), separar de la interfase todos aquellos detalles dependientes del nivel

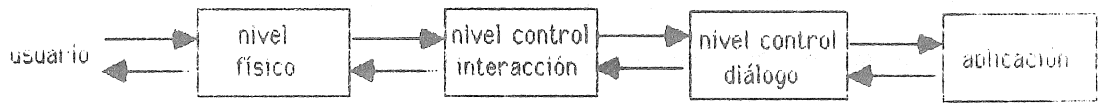
físico de tal manera que para llevarlo a otro equipo solo sea necesario redefinir esta parte. Para facilitar ésto, la interfase debe contar con una descripción completa del hardware de que dispone, librerías de rutinas que permitan la adecuada manipulación de los dispositivos y referirse en el resto de la interfase a los dispositivos en términos lógicos y nunca físicos. Esta parte se denomina el nivel físico de la interfase



A su vez hay dos partes claramente distinguibles dentro de la interfase cuando ya no depende de los detalles hardware. La primera se denomina el control de diálogo que indica la manera como debe ir moviendose la interfase del sistema segun las entradas del usuario y las posibilidades de la aplicación. Si se ve esta última simplemente como un conjunto de rutinas, la misión de este nivel es el de "oir" las ordenes del usuario, y decidir que procedimientos de la aplicación se deben llamar y en que orden, para satisfacerlo.

Por último, existe otro nivel dentro de la interfase que es el encargado de interactuar directamente con el usuario. Debe oirlo e interpretarlo, informando luego al nivel de control de diálogo lo que el usuario quiere. Este nivel se denomina el de control de interacción.

Estos dos últimos niveles son independientes entre si, en el sentido de que al control de diálogo no le interesan los detalles de cada interacción sino solo los resultados (no le interesa si el usuario escoge una opción de un menú, o si da un comando completo, o si adoptó la respuesta por defecto, o por cual dispositivo dió la respuesta), mientras que al nivel de control de interacción no le importa para que es utilizada la respuesta o la manera como será ejecutada por la aplicación.



En la bibliografía del tema, la aplicación se denomina el nivel semántico de la interfase, el nivel de control de diálogo se llama el nivel sintáctico y el nivel de control de interacción el nivel léxico. Estos nombres se encuentran inspirados en las

fases de análisis de los compiladores, y en la idea de algunos autores acerca de que una interfase no es más que un compilador de un lenguaje dinámico mediante el cual se comunica un usuario con una aplicación.

Lo importante en el modelo es la independencia entre sus diferentes niveles. Esto permite, en el momento de buscar un formalismo para expresar cada uno, escoger el más adecuado para cada tarea sin tener que pensar en una buena manera de expresar todas las características de una interfase al mismo tiempo.

Una vez definida la estructura general de una interfase se deben resolver dos problemas: el primero es decidir la forma de comunicar los niveles y el segundo es diseñar un formalismo que se ajuste a las necesidades y objetivos de cada nivel, para especificar cada una de las partes que componen la interfase. En un compilador, por ejemplo, se utilizan BNF para la parte sintáctica y expresiones regulares para la parte lexicográfica.

5. El nivel físico.

Este nivel es el encargado de garantizar la independencia entre la interfase y el hardware. Mantiene una descripción de cada uno de los dispositivos que maneja el sistema, a cada uno de los cuales le asocia un nombre mediante el cual es reconocido en el resto de la interfase. Cuenta ademas con un conjunto de rutinas que le permiten manejar cualquiera de los dispositivos presentes (dada su descripción).

Los dispositivos se dividen en dos: dispositivos de entrada y dispositivos de salida. Para cada dispositivo de entrada se debe especificar:

- nombre
- tipo: señala a que clase de dispositivo pertenece, entre las cinco posibilidades que define GKS (locator, valuator, choice, string, pick)
- rutinas de manejo del dispositivo
- características hardware especiales

Por su parte los dispositivos de salida se definen mediante los siguientes tres componentes:

- nombre
- rutinas de manejo del dispositivo
- caracteristicas hardware

La comunicación hacia el nivel de control de interacción es de la forma:

(dispositivo, contenido)

que indica el nombre del dispositivo que recibió la entrada y el contenido de la misma.

No es el objetivo de este nivel interpretar la entrada del usuario, sino simplemente recibirla por uno de sus dispositivos y transmitirla al nivel superior

6. El nivel de control de comunicación.

Este nivel es el encargado de manejar la comunicación entre el usuario y el control de diálogo. Para ésto recibe una señal de activación desde el nivel superior, se comunica con el usuario para obtener una respuesta, la interpreta y luego la manda al control de diálogo.

El nivel se encuentra conformado por un conjunto de objetos, cada uno de los cuales sabe manejar una interacción con el usuario. El que decide cuando y cual de todos los objetos se activa en un momento dado es el nivel superior.

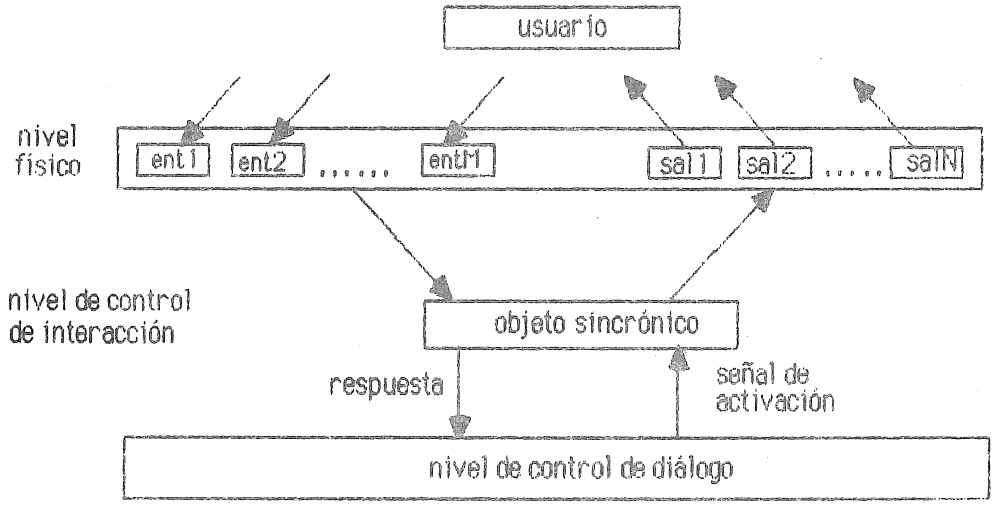
Los objetos son entonces independientes entre ellos y autosuficientes en el sentido de que saben cómo interpretar una entrada, cómo recuperarse de los errores, cómo informar al usuario que están activos, etc., descargando de todos estos detalles al control de diálogo.

Existen dos tipos de objetos en el nivel: los objetos sincrónicos y los asincrónicos. Los primeros cuando se activan toman temporalmente el control de la interfase y solo lo retornan cuando tienen una respuesta del usuario o un informe de fracaso en la interacción. Los asincrónicos cuando se activan retornan inmediatamente el control

al nivel superior y quedan dormidos hasta cuando reciben una entrada especifica del usuario. En ese momento interrumpen el control de diálogo informándole la señal dada por el usuario.

6.1. Objetos sincrónicos

Un objeto sincrónico está definido por dos partes: la interfase con los dos niveles que lo rodean y su comportamiento interno. Gráficamente la ubicación de un objeto sincrónico dentro de la estructura de la interfase sería la siguiente:



Cuando llega la señal de activación, el objeto selecciona uno o varios de los dispositivos de salida asociados e informa al usuario que se encuentra activo (presenta un menú, o le hace una pregunta, etc.). Luego coloca los dispositivos de entrada asociados al objeto en estado de alerta y por el primero que llegue una respuesta la toma, la interpreta y luego la envía al nivel de control de diálogo.

La comunicación con el nivel físico tiene entonces la siguiente forma:

(disp.salida1,contenido1, ... , disp.salidaN,contenidoN, disp.entrada1, ... , disp.entradaM)

El nivel físico sabrá, partiendo de la descripción de los dispositivos y utilizando las rutinas asociadas a cada uno, cómo manejar la comunicación directa con el usuario.

La comunicación hacia el nivel sintáctico tiene la siguiente forma:

(código de retorno, respuesta)

donde incluye la respuesta del usuario ya interpretada (ya no importa el dispositivo por el que llegó, ni la entrada física dada por el usuario, sino solamente lo que él quiere decir) y un indicador del éxito o del tipo de problema que se presentó durante la interacción con el usuario.

El comportamiento interno de un objeto es la segunda parte que se debe considerar. Dicho comportamiento es el que define la manera de reaccionar del objeto ante las diferentes situaciones que se pueden dar. Se encuentra compuesto por un conjunto de métodos que incluyen la manera de interpretar la entrada (dado el dispositivo que la recibió) y la forma de recuperarse en caso de error. Si este último no se encuentra definido en un objeto, al detectar un error simplemente lo comunica como parte del código de retorno al nivel superior. La recuperación de error puede consistir en un reintento de interacción con el usuario, la construcción de una respuesta por defecto o la misma cancelación de la ejecución de la interfase.

Existen algunas características de un objeto que solo pueden ser definidas durante la ejecución misma de la interfase. Si se trabaja por ejemplo con un objeto que presenta un menú con un número variable de opciones, dicho valor solo se conocerá en el momento de activar el objeto. Estas se denominan sus características dinámicas, y deben ir definidas como parte de la señal de activación que envía el nivel superior. Esta señal tiene por lo tanto la siguiente forma:

(nombre del objeto, características dinámicas, tiempo máximo de espera)

en la cual se incluye también un tiempo límite que se le dá al usuario para dar una entrada (timeout), y después del cual los métodos internos de comportamiento deciden como reaccionar.

6.2. Objetos asincrónicos

Un objeto asincrónico tiene una estructura un poco diferente a la expuesta anteriormente, dado que su misión no es comunicar información al nivel superior, sino control.

Existen varias tipos de señal que puede mandar el control de diálogo a un objeto asincrónico. La primera es:

(ACTIVAR, nombre del objeto, forma de interrumpir, prioridad)

mediante la cual se indica cual objeto se activa, como debe reaccionar al recibir la señal del usuario (puede dar control a una rutina del nivel semántico, o pasar a algun punto especifico del control de diálogo (ver estados seguros), etc.) y cuál es su prioridad con respecto a los demás objetos asincrónicos activos. De otra parte la señal.

(DESACTIVAR, nombre del objeto)

desactiva el correspondiente objeto asincrónico.

La definición de un objeto asincrónico consta únicamente de un posible mensaje de activación hacia el usuario (por uno o varios dispositivos de salida) y de la definición de por dónde y mediante cuál entrada se debe activar.

7. El nivel de control de diálogo

El nivel sintáctico o de control de diálogo tiene como misión relacionar las entradas del usuario con las rutinas semánticas (de la aplicación), guiando de esta manera toda la ejecución de la interfase. En este nivel, dado un contexto de ejecución y una entrada del usuario, se decide que rutinas semánticas invocar y que nuevo estado alcanzar.

A este nivel de la interfase es al cual le han dedicado más trabajo en la bibliografía que existe del tema, y casi siempre cuando se habla de interfase usuario se refieren especificamente a este nivel. Como formalismo para especificarlo utilizan BNF,

gramáticas y autómatas. En este artículo se va a presentar un formalismo de diagramas de transición aumentados, con un poder computacional equivalente al de una máquina de Turing. Esto implica que cualquier interfase que se pueda expresar en un lenguaje de programación, se puede definir en términos del modelo.

El nivel completo está compuesto por un conjunto de diagramas de transición y por un espacio de memoria compartido por todos. Uno de los diagramas de transición se conoce como el inicial, y es por éste que comienza siempre la ejecución de la interfase. Incluye además un primer nivel de ayuda que corresponde a otro de los diagramas de transición del nivel, y su misión es controlar la presentación de la ayuda más general del sistema.

Un diagrama de transición por su parte está compuesto por 7 elementos:

- nombre: forma de identificarlo desde los demás diagramas
- segundo nivel de ayuda: nombre del diagrama que contiene el segundo nivel de ayuda
- conjunto de estados (E)
- conjunto de estados seguros (S, $E \supseteq S$)
- estado inicial (I, $I \in E$)
- conjunto de estados de retorno (R, $E \supseteq R$)
- conjunto de transiciones entre los estados (T)

Un estado de un diagrama de transición representa un estado posible de ejecución de la interfase. Consta de los siguientes elementos

- nombre: forma de identificarlo desde los demás estados
- precondición: aserción que se debe cumplir al llegar al estado
- acciones sobre el estado: corresponden a llamadas de rutinas semánticas, llamadas a diagramas de transición del nivel o señales de activación a los objetos del nivel de control de interacción.
- tercer nivel de ayuda: nombre de un diagrama de transición que maneja una ayuda específica del estado
- postcondición: aserción que se debe cumplir al abandonar el estado.

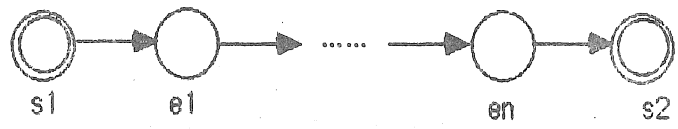
Si al llegar a un estado en un diagrama de transición, no se cumple la precondición, o si al salir no es verdad lo que se exige en la postcondición, la interfase cancela su ejecución.

Por su parte una transición indica un cambio en el estado de ejecución de la interfase causado por alguna entrada del usuario o por alguna otra situación interna. Una transición se encuentra definida por:

- prioridad: en caso de que existan dos posibles transiciones activables en un contexto dado, se toma la de mayor prioridad.
- estado de salida
- estado de llegada: estado al cual llega por la transición
- guarda de la transición: condición que debe satisfacerse para poder activar la transición.
- acción sobre la transición: corresponde a la llamada de una rutina semántica o de un diagrama de transición, que se requiere para poder cumplir la precondición del estado que se va a alcanzar.

Si en un momento dado de la ejecución no se encuentra ninguna transición activable, el control de diálogo suspende la ejecución completa de la interfase.

En cada diagrama de transición puede existir un conjunto de estados seguros, que se comportan de la siguiente manera: todo lo que se haga en el contexto entre dos estados seguros solo se confirma al llegar al segundo. Esto permite el manejo de operaciones de manera atómica (todo o nada), puesto que desde cualquier estado intermedio es posible regresar a la situación que se tenía en el último estado seguro.



Si como parte de las rutinas semánticas existen manejadores de Bases de Datos, por ejemplo, estos solo serían llamados en los estados seguros mientras en los estados intermedios se utilizaría algún tipo de mecanismo (manejar una copia, guardar todo en un archivo intermedio, etc.) para almacenar las transacciones parciales Por

efectos prácticos, todo estado de retorno se considera seguro.

Al alcanzar un diagrama de transición un estado de retorno, termina su ejecución y retorna el control al punto donde fue llamado. En todo diagrama debe existir por lo menos un estado de retorno.

Mediante la utilización de los objetos asincrónicos es posible manejar tanto los estados seguros como la ayuda del sistema, la cual tiene tres niveles: ayuda general a nivel de todo el sistema, ayuda a nivel del diagrama de transición y ayuda particular del estado en el que se encuentra.

8. El nivel semántico

El nivel semántico está compuesto por el conjunto de rutinas de la aplicación. Estas son activadas desde el nivel sintáctico el cual envía además información adicional de su memoria compartida como parámetros a las rutinas:

(nombre de la rutina, parámetros)

Dependiendo de la definición del procedimiento, algunos de estos parámetros pueden ir por referencia y por lo tanto ser modificados dentro de la rutina. No existe ningún otro tipo de comunicación entre los niveles semántico y de control de diálogo.

9. Conclusiones

El modelo que se presenta en este artículo es suficientemente general como para abarcar cualquier tipo de interfase. Existen muchos puntos que no se concretan debido a limitaciones de espacio. Se está esperando la terminación del ambiente de diseño y generación de interfases basado en el modelo, que se encuentra en desarrollo dentro del Grupo de Investigación CGI/DAC de la Universidad de los Andes, para hacer la respectiva evaluación y posible realimentación.

10. Bibliografía

[OLS83] Olsen, D., Dempsey, E., Syntax Directed Graphical Interaction, Proceedings of the SIGPLAN'83, Symposium on Programming Language Issues in Software Systems, Sigplan Not. 18, 6, pp. 112-117, Junio/1983.

[OLS84] Olsen, D., Pushdown Automata for User Interface Management, ACM Transactions on Graphics, Vol. 3, num. 3, pp. 177-203, Julio/1984.

[VIL86] Villalobos, J., Automate pour Application Interactive, Rapport de D.E.A., Institut National Polytechnique de Grenoble, Julio/1986.

[GRE85] Green, P., The University of Alberta User Interface Management System, SIGGRAPH'85, Vol. 19, num. 3, pp. 205-213, Julio/1985.

[BUX83] Buxton, W., Lamb, M., Towards a Comprehensive User Interface Management System, SIGGRAPH'85, Computer Graphics, Vol. 17, num. 3, pp. 35-42, Julio/1983.

[COU84] Coutaz, J., A Paradigm for User Interface Architecture, Department of Computer Science, Carnegie-Mellon University, Mayo/1984.

Construcción automatizada de talleres de programación: PEPIM.

Miguel SANTANA
Laboratoire de Génie Informatique - Institut IMAG
BP 68, 38402 St Martin d'Heres Cédex, Grenoble, FRANCIA

Escuela de Graduados - Pontificia Univ. Católica del Perú
Apartado 1761, Lima, PERU

RESUMEN

El ritmo acelerado de aparición de nuevos computadores estos últimos años ha creado una fuerte demanda en lo que respecta a la realización de herramientas de base para estas máquinas. Debido a esta demanda, las investigaciones sobre la automatización de la construcción de estas herramientas han cobrado un mayor auge, tanto a nivel universitario como de casas de software.

Este artículo describe el diseño y la implementación de PEPIM, un sistema de producción automática de cadenas de desarrollo de software (talleres de programación) para una máquina dada, a partir de su descripción en un lenguaje apropiado. La realización de un taller completo puede ser efectuada en pocas semanas, lo que constituye una mejora considerable en relación a la construcción manual del mismo; por otro lado, las modificaciones del taller (nuevos componentes, nuevas versiones, cambios en las especificaciones, etc.) son realizadas con mucho mayor facilidad y, por supuesto, a un costo menor.

Este proyecto ha sido realizado por el Laboratorio LGI conjuntamente con la empresa Silicone, una casa de software francesa pequeña pero sumamente dinámica. El producto elaborado representa el estado del arte actual en el área y ha despertado muchas expectativas entre los constructores de computadores, entre ellos: Texas Instruments, Motorola, NEC y Thomson.

I. INTRODUCCION.

El ritmo acelerado de aparición de nuevos computadores durante estos últimos años, ha creado una fuerte demanda en lo que respecta a la realización de herramientas de base para estas máquinas. Esta demanda se caracteriza por tres puntos principales:

a) La construcción de estas herramientas debe poder hacerse lo más rápidamente posible, debido a la importancia del factor tiempo en el mercado de computadores. En la mayor parte de los casos, la disponibilidad de las herramientas debe preceder la aparición de la máquina; de esta manera, los primeros usuarios (compañías que comercializan software, clientes importantes, etc.) pueden desarrollar con anticipación sus productos o aplicaciones.

b) La primera versión de las herramientas debe permitir un desarrollo cruzado: la programación se hará sobre una máquina diferente (llamada máquina de desarrollo) pero a destinación del nuevo procesador o computador. El desarrollo del software de base, de las primeras aplicaciones y de la mayor parte de los programas de test del hardware se hacen con la ayuda de esta versión. El término primera versión es en realidad un eufemismo para designar todas las versiones de desarrollo cruzado que serán necesarias para adaptarse a la evolución de la máquina.

c) El conjunto de herramientas, que constituyen lo que llamaremos taller de programación, debe ser lo más completo posible: compiladores de lenguajes de alto nivel, ensamblador, macro-procesador, debugger, editor de referencias (linkage editor), simulador, etc. Es raro hoy en día que los usuarios potenciales de una nueva máquina queden satisfechos con un ensamblador y un loader absoluto.

Para tratar de satisfacer esta demanda, las compañías de software se han visto obligadas a desarrollar nuevos métodos y herramientas más adaptados a la producción de este tipo de software. Sin embargo, los resultados obtenidos no han sido verdaderamente los esperados: costos muy altos, plannings demasiado largos, características técnicas pobres, número restringido de herramientas, etc.

El objetivo del proyecto PEPIM (Producción de Entornos de Programación Independientes de la Máquina) es de resolver este problema, a través de un nuevo enfoque: la automatización de la construcción de este tipo de herramientas.

El artículo propuesto presenta el diseño y la implementación de PEPIM. El primer capítulo constituye una descripción general de este sistema: orígenes, objetivos, contexto de realización, características generales. El capítulo

siguiente discute los aspectos relativos a la producción automática de los diferentes elementos del taller: informaciones necesarias, tratamiento de estas informaciones, construcción propiamente dicha. En el tercer capítulo presentamos la arquitectura general de PEPIM, describiendo de manera sucinta las características de cada uno de sus componentes. Finalmente, el cuarto capítulo analiza los resultados obtenidos con nuestro enfoque y los compara con los de productos similares.

II. DESCRIPCION GENERAL DE PEPIM.

II.1 ORIGEN DEL PROYECTO.

El Laboratorio LGI trabaja desde hace muchos años en el área de automatización de la construcción de compiladores, en el marco del proyecto GEMME [Santana 83, Santana 84, Hochain 86, Santana 86, ...]. Este proyecto nació como una respuesta al retraso del área de generación de código en compilación y ante la multiplicación de los nuevos computadores y lenguajes de programación al final de los años setenta. En ese entonces, numerosos proyectos se volcaron al estudio de este problema, entre los cuales cabe destacar: PQCC [Leverett 80], el método de generación Graham/Glanville [Graham 82, Crawford 82] y las proposiciones de Ganapathi [Ganapathi 82]. Sin embargo, pocos proyectos llegaron a resultados concretos, como fue el caso de GEMME; efectivamente, el primer prototipo de GEMME interesó bastante al Centro de Telecomunicaciones de Francia, quien decidió incorporarlo como parte del ambiente de programación CONCERTO [Concerto 86]. Gracias a esta colaboración, pudimos continuar el desarrollo de GEMME hasta llevarlo a un nivel de producto prácticamente comercial.

El proyecto PEPIM nace como una prolongación de GEMME. Este nuevo proyecto retiene las ideas más interesantes de GEMME, al mismo tiempo que mejora ciertos de sus aspectos y amplía sus objetivos. Se trata en lo esencial de aplicar las ideas desarrolladas para GEMME a un área más vasta: la construcción de todas las herramientas de base de un computador (compiladores, ensamblador, editor de referencias o linkage editor, cargador o loader, debugger, etc).

II.2 CONTEXTO DE TRABAJO.

Desde un inicio, PEPIM ha sido desarrollado en estrecha colaboración entre el Laboratorio LGI y la empresa Silicone. Este tipo de colaboración tiene la ventaja de unir los recursos propios de una empresa privada a los conocimientos (el know-how) de un laboratorio de investigación; sin embargo, también tiene su lado negativo, ligado a los imperativos de planificación y de comercialización que son los de toda empresa.

Por otro lado, cabe destacar el interés de realizar un producto de avanzada, sabiendo que va a ser comercializado y, por lo tanto, usado por un gran número de usuarios. Este hecho compensa con creces el ritmo diferente de trabajo.

II.3 OBJETIVOS.

El objetivo principal de PEPIM es la automatización total de la construcción de lo que se conoce como cadena de desarrollo o taller de programación de una máquina. Nuestra inquietud nace de la dificultad y el tiempo que toma la realización de uno de estos talleres; la automatización debe resolver ambos problemas, disminuyendo de manera importante los costos de producción.

Esta automatización debe cumplir sin embargo con ciertos imperativos:

a) Eficiencia de las herramientas obtenidas por medio de las herramientas de producción automática. Somos conscientes que esta eficiencia no será de ninguna manera igual a la de las herramientas construídas manualmente pero tampoco queremos obtener tiempos de ejecución o espacios de memoria inmensos y, por lo tanto, prohibitivos.

b) Facilidad de uso. La construcción de un nuevo taller no tiene porque ser una tarea dificultosa, aunque los usuarios sean, por lo general, especialistas. El sistema debe tratar de ayudar al usuario a todo nivel: diseño, realización, verificación, nuevas versiones, etc.

c) Modularidad del taller final. Los componentes que serán incluídos en un taller dependen de lo que ya existe en la máquina destinación (aquella para la cual se construye el taller); así, por ejemplo, puede suceder que sólo sea necesario construir un compilador C, dado que la máquina destinación ya cuenta con un ensamblador, un editor de referencias y un cargador. Por otro lado, las herramientas construídas con PEPIM deben poder ser integradas armoniosamente entre ellas y sobre todo con otras herramientas, completamente ajenas a PEPIM.

Otro objetivo importante de nuestro proyecto es la calidad del código generado a través de los compiladores construídos con PEPIM. Esta calidad debe ser igual y, en la medida de lo posible, superior a la de los compiladores de producción disponibles en el mercado; en este sentido, conservamos los mismos objetivos de GEMME.

En cuanto a las máquinas aceptadas, nos hemos limitado a la familia de máquinas a registros con posibilidades de extendernos hacia las máquinas a pila. En este aspecto, somos un poco más ambiciosos que GEMME y pensamos con ello poder cubrir casi el 99% de las máquinas existentes.

II.4 CARACTERISTICAS GENERALES.

El sistema PEPIM está compuesto de un conjunto de herramientas de producción automática de traductores, que permiten construir compiladores y un ensamblador para una máquina dada; el sistema también comporta otras herramientas, que son completamente independientes de la máquina destinación. La construcción de un nuevo taller se hace creando el (o los) traductores apropiados y ensamblándolos con el resto de herramientas. Ciertos componentes del taller pueden ser reemplazados por las herramientas del constructor de la máquina (ensamblador, editor de referencias, cargador), para guardar una mayor compatibilidad con el sistema operativo del mismo.

El sistema PEPIM ofrece además un entorno interactivo que permite la creación o la modificación de una descripción de máquina a un nivel de abstracción superior al del lenguaje de descripción propiamente dicho. Este entorno ofrece una interfase mucho más agradable, así como numerosas facilidades: formularios simplificados de descripción, deducción automática de ciertas informaciones, inicialización a partir de la descripción de alguna otra máquina, generación de programas de test, etc.

El conjunto de herramientas que pueden ser construídas por medio de PEPIM es bastante completo: ensamblador, compilador e intérprete C, compilador Pascal (aún no disponible), debugger de alto nivel, editor de referencias, cargador; es decir, prácticamente todo el software de base para una nueva máquina o procesador. Todas estas herramientas se integran armoniosamente y constituyen un verdadero taller de programación. Cabe indicar que el usuario puede construir únicamente los componentes que le faltan y enriquecer así su taller de programación: las técnicas desarrolladas permiten tomar en cuenta las características de los elementos existentes e inclusive su uso al interior de las herramientas generadas por PEPIM; así, por ejemplo, un compilador C generado por PEPIM producirá un programa objeto compatible con el editor de referencias del sistema operativo de la máquina.

III. CONSTRUCCION AUTOMATIZADA DE TALLERES.

III.1 ANALISIS DEL PROBLEMA.

La idea de base del proyecto consiste en tratar de separar las informaciones que dependen de la máquina destinación y aquellas que son independientes de la misma. De esta manera, las herramientas pueden ser descompuestas en dos partes: una común a todas las máquinas (parte algorítmica) y otra específica a cada máquina (parte datos). Todas las características de la máquina de destinación serán reunidas en un conjunto de variables y de tablas, que conformarán la parte datos de las herramientas; la

construcción de una nueva herramienta necesitará por lo tanto la escritura de esta segunda parte.

Este enfoque permite reducir de manera apreciable los costos de producción, puesto que una parte importante de las herramientas es realizada una sola vez.

Sin embargo, la construcción manual de la parte datos constituye una tarea bastante larga y fastidiosa, además de poco fiable; es por ello que hemos preferido que esta tarea sea efectuada automáticamente por PEPIM, a partir de una descripción de la máquina destinación en un lenguaje apropiado.

Esta descripción contiene un cuerpo común al conjunto de herramientas del taller y una serie de anexos propios a cada una de las herramientas; cabe indicar que ciertas herramientas no poseen un anexo propio, sea porque las informaciones del cuerpo de la descripción les son suficientes, sea porque son completamente independientes de la máquina de destinación; es el caso del analizador C, de su intérprete, del linkage editor, etc. En cambio, todos los traductores poseen un anexo especifico.

III.2 CONSTRUCCION DE UN ENSAMBLADOR.

La construcción de un ensamblador necesita principalmente dos tipos de informaciones:

a) La sintaxis del lenguaje ensamblador. Una parte de esta sintaxis es fija e idéntica para todos los ensambladores construídos con PEPIM; se trata esencialmente de la representación de las unidades lexicales (constantes, identificadores, comentarios, etc.) y de la disposición de las informaciones en una línea fuente. Otra parte de las informaciones debe ser descrita, pues depende de la máquina destinación:
   - representación de los modos de direccionamiento,
   - mnémonicos de las instrucciones,
   - tipos de secciones y como van a ser direccionadas.

   Por otro lado, el usuario tiene la posibilidad de definir ciertas directivas o completar algunas otras; sin embargo, la mayor parte de las directivas son fijas.

b) El formato de cada instrucción, el cual incluye:
   - el número de operandos,
   - los modos de direccionamiento que acepta la instrucción para cada operando,
   - el formato binario de la instrucción.

En ciertos casos, un tercer tipo de información puede ser necesario: el formato bajo el cual debe ser generado el código objeto; esto sucede cuando se quiere usar un editor de referencias que ya existe y no aquel que es propio a PEPIM.

Todas estas informaciones forman parte de la descripción de la máquina destinación. Sin embargo, algunas de ellas son comunes a la generación del ensamblador y de los compiladores: modos de direccionamiento, familias de modos y sintaxis del lenguaje de ensamblaje.

Estas informaciones son transformadas por PEPIM, luego de una verificación, en tablas que serán incorporadas a un esqueleto de ensamblador para dar nacimiento al ensamblador deseado. Estas tablas son generadas bajo la forma de un programa C normal y, luego de ser compiladas, linkeditadas con el resto del ensamblador.

III.3 CONSTRUCCION DE UN COMPILADOR.

La principal dificultad para la construcción automática de un compilador reside en la parte generación de código del mismo. Efectivamente, el uso de herramientas de construcción automática de analizadores se ha vuelto banal estos últimos años, dada la fiabilidad y la facilidad de uso alcanzadas por estas herramientas [Johnson 75, Aho 86]; por el contrario, las herramientas e inclusive los métodos de construcción automática de generadores de código son más bien escasos. Dado que PEPIM constituye una prolongación del proyecto GEMME, fue natural que escogiéramos el método y las herramientas desarrollados en éste [Santana 86].

Las características principales del método de generación de GEMME son las siguientes:

a) Se trata de una solución integrada, que toma en cuenta todos los aspectos de la generación de código. Sin embargo, la selección de instrucciones ha sido privilegiada por el hecho de constituir la fuente de optimizaciones más rica.

b) La selección de instrucciones está basada en una búsqueda exhaustiva; por esta razón, el código generado es siempre el mejor a nivel local. Por otro lado, los algoritmos correspondientes son más simples y sobre todo más generales.

c) Todos los datos relativos a la máquina destinación son conservados en tablas; de esta manera, los mismos algoritmos pueden ser usados para máquinas diferentes: basta con cambiar las tablas.

d) En razón de su fuerte interacción, las diferentes tareas de la generación son hechas simultáneamente (asignación de registros, implantación de variables, traducción de tipos, administración de temporarios, selección de instrucciones, etc).

El generador de código resultante puede ser visto como un intérprete dirigido por las diferentes tablas de descripción de la máquina destinación:
- tabla de objetos de la máquina (representación de los datos, tipos existentes, registros, descripción de la memoria, etc),
- tabla de modos de direccionamiento y de sus clases,
- tabla de operaciones que indica las diferentes maneras de traducir cada operación del lenguaje fuente,
- sintaxis del lenguaje ensamblador que debe ser generado.

El generador descompone el programa en curso de compilación en operaciones simples, que son sometidas al mecanismo de selección de instrucciones, elemento central de nuestro método de generación. Este mecanismo efectúa un recorrido exhaustivo de las instrucciones que pueden ser usadas para traducir la operación y genera la mejor secuencia de instrucciones que encuentra; este recorrido exhaustivo nos permite aprovechar al máximo los modos de direccionamiento y el juego de instrucciones de la máquina.

Como se puede observar, el generador es tan sólo un esqueleto, que tiene que ser completado con las tablas indicadas para obtener un verdadero generador de código. La construcción se hace exactamente de la misma manera que en el caso de un ensamblador: la descripción es transformada en tablas C, que luego de ser compiladas son linkeditadas con el esqueleto.

Gran parte de las informaciones necesarias para construir un compilador son compartidas con el ensamblador.

III.4 CONSTRUCCION DE OTROS COMPONENTES.

Los otros componentes de PEPIM pueden ser considerados independientes de la máquina destinación. Así, por ejemplo, el analizador C necesita muy pocas informaciones sobre la máquina, al igual que el intérprete C: tamaño de cada tipo de base, disposición de los bits en una palabra. Por su lado, el editor de referencias y el cargador de PEPIM son completamente independientes de la máquina destinación; si el usuario decide usar otro editor de referencias, el formato objeto que éste usa deberá ser descrito, para que el ensamblador pueda hacer uso de él.

Todos estas herramientas han sido realizadas de manera a ser totalmente independientes de la máquina destinación, sea retrasando el tratamiento correspondiente hasta una etapa posterior, sea tomando ciertas hipótesis de funcionamiento. Por consiguiente, estas herramientas son las mismas en todos los talleres construídos con PEPIM. En realidad, existen ciertas diferencias, debidas en su mayor parte a problemas de interface, tanto con el usuario como con el sistema operativo; sin embargo, estas diferencias no son conceptuales y, por lo tanto, sin importancia.

IV. ARQUITECTURA DE PEPIM.

IV.1 ORGANIZACION GENERAL.

El sistema PEPIM está organizado en dos niveles:

a) Generación de herramientas, que permite construir una herramienta dada, a partir de una descripción de la máquina destinación y de un esqueleto o parte algorítmica de esta herramienta (ver capítulo III).

b) Taller de programación, constituído por las herramientas generadas por el nivel anterior y por aquellas que son independientes de la máquina destinación.

Esta organización está ilustrada por la figura 1.

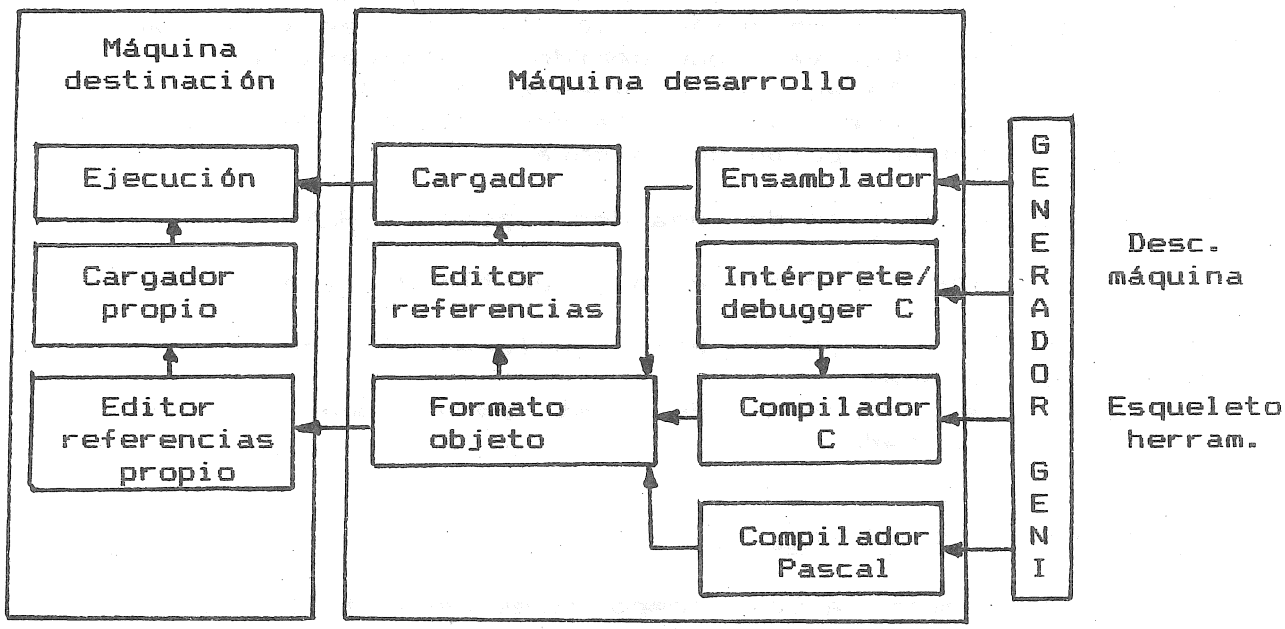


Figura 1. Organización general de PEPIM.

IV.2 SISTEMA DE AYUDA PARA CONSTRUIR UN TALLER.

La descripción de la máquina destinación puede ser hecha en un lenguaje que hemos definido para este efecto (Lenguaje de Descripción o LD). Sin embargo, esta tarea puede resultar sumamente tediosa, por la cantidad de modos de direccionamiento e instrucciones de una máquina, y poco fiable, por el gran número de opciones e informaciones puntuales que requiere nuestro sistema. Es por ésto que decidimos tomar un enfoque radicalmente opuesto, que consiste en ofrecer un sistema interactivo de ayuda para la construcción del taller: el sistema GENI (GENerador Interactivo).

GENI ofrece numerosas facilidades para la confección de una descripción:
- formularios de descripción simplificados,
- menús para escoger las opciones,
- valores por defecto o deducidos a partir de otros (algunos tienen que ser confirmados por el usuario),
- jerarquización de la descripción (acceso secuencial y/o directo a las rúbricas),
- verificación de la coherencia entre las diferentes rúbricas,
- indicación de las partes incompletas,

etc.
Durante el trabajo de edición, GENI hace uso de la división en cuerpo principal y anexos de la descripción, tanto para guiar al usuario como para pedirle sólo las informaciones estrictamente necesarias.

Al final de cada sesión, GENI produce el programa equivalente en lenguaje LD, representación bajo la cual es almacenada la descripción. Cabe observar que debido a esta característica, el usuario puede trabajar directamente en LD, si lo desea; igualmente, puede hacer uso de la descripción de otra máquina para comenzar la descripción de la suya.

Aparte de la edición de una descripción, GENI ofrece otros servicios:
- construcción de una herramienta, a partir de la descripción en curso,
- listado de la descripción bajo su forma LD, con una tabla de referencias cruzadas,
- generación de programas de test para poner a punto la descripción realizada.

IV.3 COMPONENTES DE UN TALLER.

La figura 2 muestra los componentes de un taller que pueden ser construídos a través de PEPIM. Esta misma figura pone en relieve las relaciones entre estos componentes, así como las diferentes representaciones que puede tomar un programa.

Es importante destacar la estrecha relación que existe entre el compilador y el intérprete/debugger C. Estas dos herramientas están articuladas alrededor de una representación intermediaria común, que permite tanto la interpretación del programa como su traducción en lenguaje assembler. El intérprete permite ejecutar el programa del usuario, ayudándolo en su puesta a punto con mecanismos como:
- visualización de la función en curso de ejecución e indicación de la instrucción ejecutada,
- trazado de variables o de funciones,
- breakpoints asociados a instrucciones o funciones, con posibilidad de ser activados únicamente bajo ciertas condiciones (contadores, cierto valor de una expresión, identidad de la función que hace el llamado, etc),

- ejecución paso a paso,
- evaluación de expresiones y funciones desde el lenguaje de comandos,
- acceso a la pila del programa C,

etc.

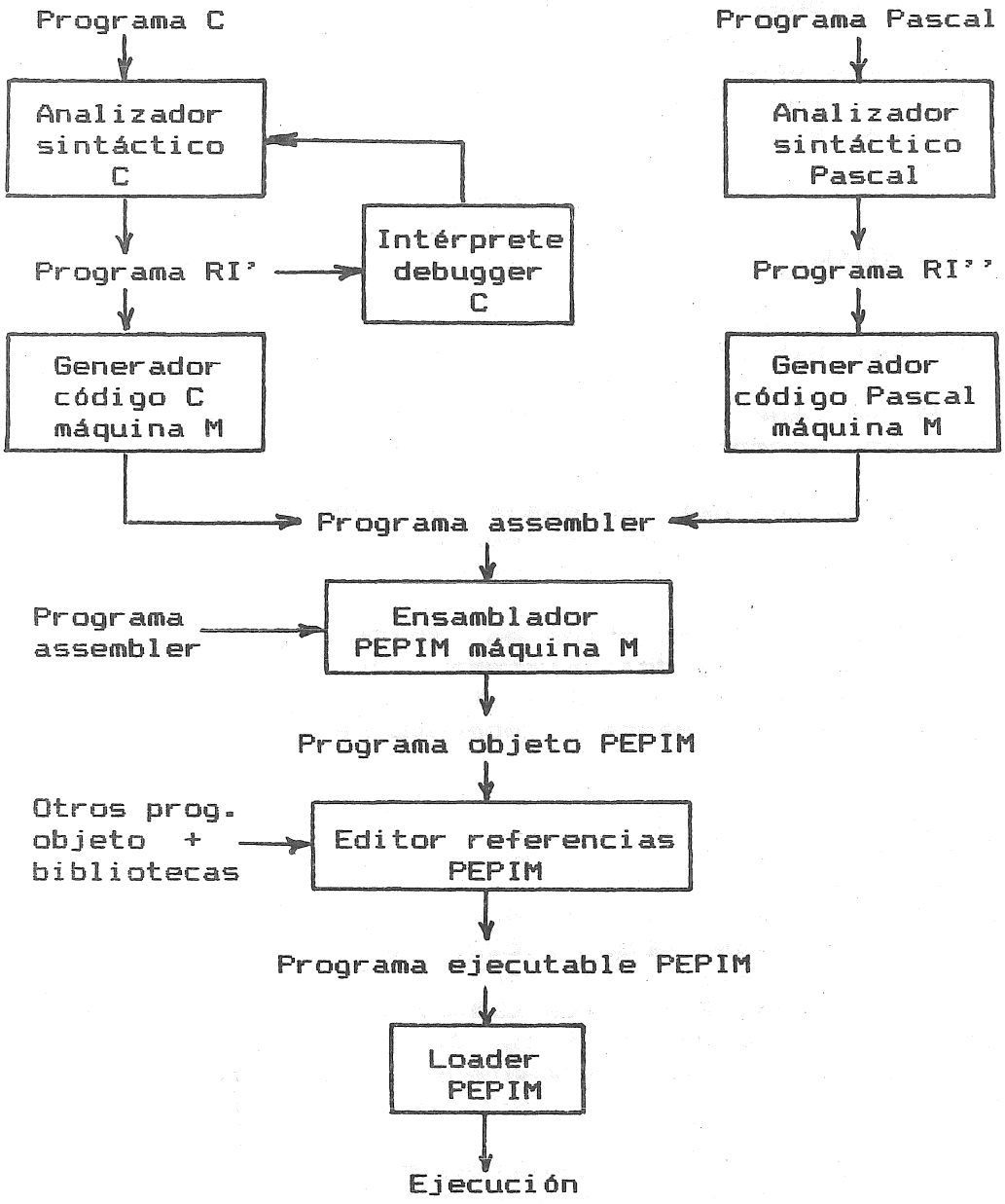


Figura 2. Taller de programación.

Como se puede observar, el intérprete/debugger trabaja a nivel de programa fuente, lo que le permite efectuar todo tipo de verificaciones sobre el comportamiento del programa: control de tipos, apuntadores con direcciones incorrectas, pasaje de parámetros incorrecto, etc. Por otro lado, el usuario manipula directamente los objetos que él ha definido (variables, instrucciones, funciones), sin necesidad de conocer ni direcciones ni el lenguaje assembler o binario de la máquina destinación. Cabe destacar igualmente el cuidado con el que ha sido realizada la interfase de esta herramienta, dada su importancia en el taller.

V. RESULTADOS OBTENIDOS.

V.1 ESTADO ACTUAL DEL SISTEMA.

El proyecto PEPIM fue iniciado en el mes de Setiembre de 1985, cuando se comenzaron a realizar las primeras especificaciones: arquitectura del sistema, generador de ensambladores, definición del lenguaje LD. Desde entonces, una gran parte del sistema ha sido implementado, aunque aún faltan ciertos elementos.

Nuestra estrategia ha consistido en desarrollar primero los componentes más importantes del sistema, de manera a poder mostrarlos a los clientes potenciales, y a incrementar su número y calidad progresivamente. Hasta el momento han sido realizados y entregados:

- ensamblador,
- analizador sintáctico C,
- intérprete/debugger C,
- generador de código C,
- editor de referencias (link) y cargador (loader),
- compilador del lenguaje LD, incluyendo los anexos que corresponden a las herramientas enumeradas.

Por otro lado, el sistema GENI está siendo desarrollado en este momento y debe ser entregado a fines del año 1987; una parte importante de GENI funciona actualmente y es usada internamente por el LGI y Silicone.

Varias herramientas han sido construídas con los componentes disponibles: ensambladores para el 68000, el 8086 y el 6502, un compilador C para el procesador Texas C25 y para el 68020, etc.

V.2 PORTABILIDAD DEL SISTEMA.

La portabilidad es un aspecto importante en un sistema como PEPIM, puesto que se trata de un sistema de desarrollo cruzado que tiene que ser instalado en diferentes máquinas de desarrollo, según la evolución del mercado. Es por esta razón

que nuestra implementación ha sido cuidadosamente realizada, con el objetivo de alcanzar la mayor portabilidad posible.

PEPIM ha sido implementado en C, por tratarse del lenguaje más portable de nuestros días. Por otro lado, hemos tratado de evitar todas las construcciones C que puedan resultar conflictivas en un transporte y previsto los problemas inevitables de representación de datos.

El sistema ha sido desarrollado en un VAX 785 bajo el sistema UNIX Berkeley 4.3; desde hace algunos meses, migramos hacia un MicroVAX II bajo el sistema VMS, sin mayores problemas. Para cerciorarnos de la portabilidad de nuestro sistema, hemos efectuado algunas experiencias de transporte de componentes; entre éstas, cabe destacar el transporte de la parte de generación de ensambladores hacía tres máquinas diferentes (Micromega, Sun 3 y PC-IBM), sin encontrar dificultades mayores.

El primer transporte del conjunto del sistema será hacia los PC-IBM y compatibles, a partir del mes de Octubre de 1987.

V.3 PERFORMANCES ALCANZADAS.

La eficiencia de las herramientas construídas con PEPIM constituye uno de nuestros principales objetivos: ésta debe mantenerse dentro de lo razonable y de ninguna manera constituir un freno al uso de las herramientas.

Para verificar el cumplimiento de este objetivo, hemos realizado dos tipos de comparaciones:

a) Ensambladores generados por PEPIM contra ensambladores entregados por el constructor mismo. En este caso, las cifras son bastante dispares pero permiten formarse una idea: los ensambladores UNIX (as) son en promedio dos veces más rápidos que los nuestros, mientras que el MASM o el ASM-86 del PC-IBM son prácticamente equivalentes a nuestro ensamblador 8086.

b) Generador de código PEPIM contra generador de código GEMME. Dado que PEPIM constituye en cierto modo una mejor implementación de GEMME, esta comparación nos permite observar la influencia de las mejoras introducidas así como del cambio de lenguaje de desarrollo (C en vez de Lisp); por otro lado, las medidas tomadas para evaluar GEMME nos permiten hacer comparaciones indirectas con otros compiladores. Nuestras primeras medidas indican un alza de aproximadamente 50% de la eficiencia respecto de GEMME; cabe indicar que las evaluaciones de éste señalaban un tiempo entre dos a tres veces mayor al de los compiladores de producción.

VI. CONCLUSIONES.

El sistema presentado en este artículo representa una respuesta a las necesidades crecientes en el área del desarrollo de software de base. El diseño de este sistema incorpora técnicas bastante avanzadas y no por ello pierde su calidad de producto utilizable industrialmente.

El anuncio de PEPIM atrajo el interés de numerosos constructores de computadores, algunos de los cuales ya han concretizado este interés a través de contratos de colaboración o de desarrollo. Entre estas compañías podemos citar Texas Europe (compilador C para su procesador C25), Thomson (taller completo para su nuevo procesador en tratamiento de señales) y SFENA (compilador C para un procesador especializado en la aviación). Otras compañías están negociando en estos momentos contratos similares (NEC y Motorola).

Diferentes desarrollos han sido previstos durante el año 1988, para consolidar y completar nuestro sistema. La primera prioridad ha sido dada a la optimización y mejora del sistema actual: mayor eficiencia, transporte hacia otras máquinas de desarrollo (PC-IBM, Sun y Apollo), mejora de la interfase de GENI, etc. Después de esta etapa, nos gustaría agregar nuevas herramientas:

- compilador Pascal,
- intérprete/debugger Pascal,
- editores sintácticos C y Pascal,

y, en un futuro más lejano, un generador de simuladores.

BIBLIOGRAFIA

[Aho 86]
A. Aho, R. Sethi, J. D. Ullman, "Compilers. Principles, techniques and tools", Addison-Wesley, 1986

[Concerto 86]
"CONCERTO: Atelier de logiciel. Présentation technique", CNET Lannion, Perros-Guirec, Febrero 1986.

[Coussediere 85]
L. Coussedière, M. Santana, "Atelier logiciel: Spécifications générales", TR DC-068-01, LGI/Silicone, Grenoble, Octubre 1985.

[Crawford 82]
J. Crawford, "Engineering a production code generator", Sigplan 82 Symposium on Compiler Construction, Boston, Junio 1982, p. 205-215.

[Ganapathi 82]
M. Ganapathi, C. Fischer, "Description driven code generation with attribute grammars", 9th Annual Symposium on Principles of Programming Languages, Albuquerque, Enero 1982, p. 108-119

[Graham 82]
S.L. Graham, R.R.Henry, R.A. Schulman, "An experiment in table driven code generation", Sigplan 82 Symposium on Compiler Construction, Boston, Junio 1982, p. 32-43

[Hochain 86]
J.C. Hochain, M. Santana, "Génération de code multi-machine", Jornadas CONCERTO, Perros-Guirec, Febrero 1986

[Johnson 75]
S.C. Johnson, "YACC: yet another compiler-compiler", TR 32, Computing Science, Bell Laboratories, Murray Hill, 1975

[Leverett 80]
B.W. Leverett y al, "An overview of the Production Quality Compiler-Compiler project", IEEE Computer 13-8, Agosto 1980, p. 38-49

[Santana 83]
M. Santana, "Un système de production automatique de générateurs de code", Facultad de Informática, Universidad de Grenoble, Diciembre 1983

[Santana 84]
M. Santana, J.C. Hochain, "Un sistema de producción automática de generadores de código", IV International Conference on Computer Science, Santiago de Chile, Junio 1984

[Santana 86]
M. Santana, "Notre expérience avec le générateur de code multicible GEMME", 2das Jornadas JISI, Túnez, Abril 1986

MESTRA: UMA METODOLOGIA DE ESCRITA DE DOCUMENTAÇÃO INTERNA TRANSPORTÁVEL

CAMILO DE LELIS GONDIM MEDEIROS

UNIVERSIDADE FEDERAL DA PARAÍBA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
AV. APRÍGIO VELOSO, S/N - BODOCONGÓ
58.100 - CAMPINA GRANDE - PB - BRASIL

RESUMO: Este trabalho apresenta uma Metodologia de Programação voltada para a escrita de comentários transportáveis a serem incluídos em programas que serão submetidos a um Sistema Transformacional para a derivação de suas várias versões. Aqui são também apresentados os conceitos de comentários constantes e comentários variáveis. Um pré-processador para conversão automática dos comentários variáveis é delineado.

1. INTRODUÇÃO

Embora o desenvolvimento de Software de Análise Numérica seja feito tradicionalmente utilizando a Linguagem de Programação FORTRAN, a sua transportabilidade nem sempre é assegurada. Para solucionar os problemas de portabilidade, uma solução frequentemente adotada consiste em efetuar manualmente todas as adaptações até que o software produza os resultados desejados. Uma outra solução, modernamente adotada, para o mesmo problema, é a utilização dos Sistemas Transformacionais que, em geral, transformam programas FORTRAN previamente padronizados.

A redução dos custos de desenvolvimento e o grau de confiabilidade do software assim produzido são os dois aspectos que mais encorajam a utilização dos Sistemas Transformacionais. Para que um Sistema Transformacional tenha ampla aceitação é aconselhável que ele exija o mínimo possível de padronização na escrita dos programas que serão transformados. Alterar automaticamente comentários, fornecer os programas transformados com uma boa formatação, mudar a precisão da aritmética utilizada e efetuar otimizações em programas fonte, são outras características importantes dos Sistemas Transformacionais.

Enquanto que os Sistemas Transformacionais foram amplamente abordados por [1], [3], [4], [9], [10], [12] e [22]; Transformações e Transportabilidade de programas FORTRAN foram discutidos por [2], [5]-[8], [11], [13], [14], [16]-[19], [23], [24], [26] e [27]; a padronização e a documentação de programas foram abordados por [15], [25] e [28], entre outros: a portabilidade da documentação interna foi superficialmente discutida em [20] e será tratada, de maneira mais detalhada, no presente trabalho. A seguir apresentamos uma disciplina de programação, que chamamos de MESTRA (Metodologia de Escrita de Documentação Interna Transportável), que visa facilitar a obtenção de programas transportáveis inclusive quanto à documentação interna.

As proposições aqui efetuadas levam em consideração que a linguagem FORTRAN é utilizada no desenvolvimento dos programas, podendo, no entanto, ser adaptadas para outras linguagens de programação de alto nível sem maiores problemas, e supõem a existência de um pré-processador para alterar os comentários existentes nas unidades de programa.

A utilização dos comentários na forma em que estamos sugerindo é justificável quando várias versões devem ser produzidas a partir de uma mesma unidade de programa, quer por razões de mudança da precisão aritmética utilizada, quer por razões de mudança de ambiente de computação ou até mesmo pela introdução de otimizações. Esta forma de comentários está sendo utilizada atualmente na Biblioteca Transportável de Análise Numérica (BITAN) da Universidade Federal da Paraíba, em Campina Grande, onde existe um pré-processador chamado Sistema Transformador de Programas FORTRAN (TRAPO) que, além de desempenhar outras funções, altera automaticamente os comentários das unidades de programa, quando necessário (veja [20] e [21] para maiores detalhes).

2. UMA PADRONIZAÇÃO PARA ESCRITA DOS COMENTÁRIOS.

Na MESTRA são utilizados dois tipos e duas classes de comentários os quais descrevemos a seguir. Quando possível todos estes comentários devem ser incluídos, na forma descrita, nas unidades de programa.

2.1. CLASSIFICAÇÃO DOS COMENTÁRIOS

Há duas classes de comentários em programas: os Comentários de Prólogo e os Comentários Explicativos (veja [28]). O conteúdo destes comentários fica a cargo da livre escolha do programador. Diferente dos comentários tradicionalmente utilizados, cujas informações são irrelevantes em tempo de compilação (e por isso são desprezados pelos compiladores), sugerimos a utilização de comentários de dois tipos: os comentários constantes e os comentários variáveis (veja [20]).

As Classes de Comentários.

Cada unidade de programa da biblioteca contém alguns comentários no início para explicar o que ele faz. Estes comentários são chamados de Comentários de Prólogo. A segunda classe de comentários, os Comentários Explicativos, são inseridos na unidade de programa para explicar qualquer código fonte cujo entendimento não seja possível com a sua simples leitura.

Os Tipos de Comentários.

Os comentários constantes são escritos da maneira convencional, começando pelo caráter 'C' e contendo informações que não mudam para as várias versões de uma mesma unidade de programa. Os comentários variáveis são comentários especiais cujas informações variam de versão para versão de uma mesma unidade de programa e, por isso, devem ser considerados pelo Sistema ao efetuar qualquer tipo de transformação.

2.2. CONSIDERAÇÕES SOBRE O CONTEÚDO E A FORMA DOS COMENTÁRIOS

Visto que a documentação interna das unidades de programa deve ser escrita levando em consideração sua transportabilidade, propomos algumas padronizações sobre a forma de escrevê-la para que a mesma possa ser transportada com o mínimo esforço. Os programadores que estiverem escrevendo software transportável deverão ter sempre em mente o princípio básico de que deverão ser evitadas quaisquer referências a nomes de variáveis, de equipamentos, de precisão, ou de quaisquer outras entidades que sirvam, de alguma forma, para associar a unidade de programa a uma versão específica de unidade de programa, à exceção dos lugares destinados a tal finalidade.

Comentários Explicativos.

A quantidade e o conteúdo dos comentários são dois

aspectos bastante importantes. Recomenda-se adotar a sugestão de Tassel [28] e utilizar pelo menos um Comentário Explicativo para cada dez (10) linhas de código fonte.

Quanto ao conteúdo, recomenda-se ao programador que assuma a familiaridade do leitor com a linguagem FORTRAN e, portanto, os comentários devem explicar a finalidade de um grupo de comandos do programa, e não, descrever a operação dos comandos. Por exemplo, antes do comando:

```
IF ( Q.NE.0 ) GO TO 320,
```

um comentário como:

```
C     DESVIO SE Q DIFERENTE DE ZERO
```

não é um bom comentário pois tenta explicar a sintaxe do comando em vez de informar por que um desvio deve ser usado.

Por outro lado, o comentário:

```
C
C          QUANDO O PARÂMETRO DE MARQUARDT É ZERO ATRIBUI-LHE
C          O MENOR DOS VALORES DO TRAÇO E NORMA INFINITA ( MÁ
C          XIMA  SOMA DOS ELEMENTOS DA LINHA) DA MATRIZ  NOR-
C          MAL.
C
```

tirado de uma unidade de programa da BITAN, onde aparece antes do comando acima, é um bom comentário pois informa por que um desvio deve ser usado.

Comentários de Prólogo

O programador deve incluir nos Comentários de Prólogo, no mínimo, como sugerido por Tassel [28], as seguintes informações, observadas as convenções de padronização que incluímos:

(1) Uma descrição do que a unidade de programa faz, escrita em comentários convencionais da linguagem FORTRAN;

(2) Versão: o programador deve incluir três linhas de comentários variáveis para identificar a versão da unidade de programa: a primeira linha contém "C& VERSÃO:", a segunda contém "C& EQUIPAMENTO:" seguido da identificação do equipamento utilizado, e a última contém "C& PRECISÃO: " seguido da precisão usada na versão. Por questões de eficiência de execução decidiu-se pela utilização de comentários variáveis, iniciados por "C&", o que dispensa o pré-processador de analisar um grande número de comentários (os comentários convencionais e que são maioria, quase sempre). Quando a unidade de programa for submetida ao TRAPO para a geração de novas versões, o Sistema se encarrega de preencher os valores referentes a " EQUIPAMENTO" e"PRECISÃO" com informações obtidas do "menu " preenchido pelo usuário;

(3) Utilização: como chamar ou utilizar a unidade de programa. Deve-se ser incluída uma linha contendo "C& UTILIZAÇÃO:", seguida do exemplo de utilização, que deve ser colocado em uma nova linha de comentário variável começando pelos caracteres "C&". Como esta é uma linha especial de comentário, caso necessite ser continuada, a linha de continuação também deverá começar pelos caracteres"C&";

(4) Uma lista e descrição de todos os parâmetros. A lista deve ser iniciada por uma linha contendo "C& PARÂMETROS:" e deve conter, para cada parâmetro: uma linha começando pelos caracteres "C&" seguidos do parâmetro e do seu tipo(acompanhado da dimensão ou dimensões, se aplicável) separados por pelo menos um espaço em branço um do outro, seguida de uma ou mais linha(s) de comentários descrevendo o parâmetro. A lista descreve inicialmente os parâmetros que são passados para o subprograma e a seguir, os parâmetros que são retornados do subprograma. Na chamada, os parâmetros obedecem também a este sequenciamento. Su

gere-se que a descrição de cada parâmetro seja iniciada na mesma posição de tabulação que aquela usada para iniciar o tipo do parâmetro;

(5) Uma lista dos subprogramas utilizados. A lista deve ser iniciada por uma linha contando " C& SUBPROGRAMAS:". Este comentário deve ser iniciado pelos caracteres "C& e deve conter o nome e a classe de todos os subprogramas utilizados na unidade de programa (um por linha). Adicionalmente, cada nome de suprogramas pode vir acompanhado, além da classe, por uma breve descrição do que faz o subprograma, nesta ordem. Primeiro descrever os subprogramas externos e depois os intrínsecos que foram utilizados;

(6) O nome de qualquer método científico utilizado, acompanhado de referência sobre onde mais informações possam ser encontradas;

(7) Alguma indicação do tempo requerido para execução;

(8) Tamanho da unidade de programa:
- .número de comandos
- .memória requerida;

(9) Requisitos especiais de operação;

(10) Autor;

(11) Data em que a unidade de programa foi escrita.

Todas estas informações devem ser colocadas diretamente no corpo da unidade de programa para que possam ser encontradas facilmente. Assim, a unidade de programa conterá a sua própria documentação.

3. PRÉ-PROCESSADOR PARA CONVERSÃO DA DOCUMENTAÇÃO INTERNA.

Os comentários de prólogo de tipos 2, 3, 4 e 5 são comentários variáveis (começam com "C& ") e são analisados pelo pré-processador ao efetuar a conversão da documentação interna. Os demais comentários, inclusive os explicativos, são escritos da maneira convencional (começando com o caráter C) com a pequena restrição de não ter o C inicial seguido do caráter " & ", e portanto, não são levados em consideração ao se efetuar a conversão.

Se por alguma razão o programador sentir a necessidade de escrever nos comentários o nome de entidades que estejam relacionadas com uma versão específica de unidade de programa , isto só será permitido para os comentários de prólogo de tipos (3), (4) e (5), que são comentários variáveis, com a imposição de que as entidades sejam analisadas quanto à possibilidade de conversão. Em última instância, se o programador tiver que efetuar referência, em um comentário explicativo, a nomes de entidades que estejam relacionadas com uma versão específica de unidade de programa, este comentário deverá ser um comentário variável (portanto, será iniciado por "C &") e o(s) nome(s) da(s) referida(s) entidade(s) dever(á) (ão) vir antecedido(s) pelo caráter " & " em cada referência, para que o Sistema possa identificar as transformações a serem efetuadas.

Em todos estes comentários variáveis o Sistema tenta identificar a entidade a ser convertida (vem antecedida por " &") e atua de acordo com as ações especificadas na tabela 1. Mas, qualquer que seja o caso, o Sistema deixa o comentário iniciando por "C&", e deixa o " & " antes das entidades, para que novas versões da unidade de programa possam ser produzidas a partir da que está sendo preparada, se vierem a ser requeridas. Isto possibilita a reversibilidade de conversão da documentação interna para uma forma "equivalente", sempre que necessário. Um módulo do Sistema elimina todas as ocorrências de ' & ', do texto fonte, quando requerido.

4. CONCLUSÃO

Quando várias versões são automaticamente produzidas para uma mesma unidade de programa, a partir da mesma versão mestre, por um programa de computador, torna-se particularmente importante que a sua documentação seja também automaticamente modificada. A conversão automática da documentação interna implica, além de confiabilidade, em redução nos custos de produção de programas desenvolvidos utilizando Programação Transformacional (definida em [22]).Unidades de Programa com documentação interna transportável podem ser obtidas seguindo a nova Metodologia de Programação ora proposta, desde que se tenha um pré-processador para efetuar as devidas atualizações.

5. BIBLIOGRAFIA

[1] AIRD, Thomas J. - The IMSL FORTRAN Converter:An Aproach to Solving Portability Problems, **Portability of Numerical Software**, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 369-388.

[2] AIRD, Thomas J.; Battiste, Edward L.; Gregory,Walton C. - Portability of Mathematical Software Coded in FORTRAN, ACM **Trans. Math. Software**, 3 (1977),pp.133-127.

[3] BENTLEY, J. & Ford, Brian - On the Enhancement of Portability Within the NAG Project - A Statistical Survey, **Portability of Numerical Software**, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976,pp.505-528.

[4] BOYLE, James M. - Mathematical Software Transportability Systems - Have the Variations a Theme?, **Portability of Numerical Software**, W. Cowell (ed.), Oak Brook , Illinois, Jun/1976, pp. 305-360.

[5] BOYLE, J. & Matz, M. - Automating Multiple Program Realizations, **Symposium on Computer Software Engineering**, Polytechnic Institute of New York - Apr/1976,pp. 421-456.

[6] BROWN, W.S. & Hall. A.D. - FORTRAN Portability via Models and Tools, **Portability of Numerical Software**,W.Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp.158-164.

[7] CODY, W. J. - Machine Parameters for Numerical Analysis, **Portability of Numerical Software**, W. Cowell (ed.)Oak Brook, Illinois, Jun/1976, pp. 49-67.

[8] DEKKER, T.J. - Machine Requirements for Reliable, Portable Software, **Portability of Numerical Software**, W. Cowell (ed.) Oak Brook, Illinois, Jun/1976, pp.22 - 36.

[9] DRITZ, Kenneth W. - Multiple Program Realizations Using the TAMPR System, **Portability of Numerical Software**, W. Cowell (ed.)O.Brook, Illinois, Jun/1976, pp. 405 - 423.

[10] DU CROZ, J. J.; Hague, S.J.; Siemieniuch, J. L.-Aids to Portability within The NAG Project, **Portability of Numerical Software**, W. Cowell (ed.), Oak Brook,Illinois, Jun/1976, pp. 389-404.

[11] FORD,Brian - Preparing Conventions for Parameters for Transportable Numerical Software, **Portability of Numerical Software**, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 68-91.

[12] FORD, Brian; Bentley, J.; Du Croz, J.J.; Hague, S. J. - The NAG Library 'Machine', **Software - Practice and Experience**, 9 (1979), pp. 65-72.

[13] FORD, Brian & Sayers, D.K. - Developing a Single Numerical Algorithms Libray for Different Machine Ranges , ACM Trans. **Math. Software**, 2 (1976),pp. 115-131.

[14] HAGUE, Stephen J. & Ford, Brian - Portability - Prediction and Correction, **Software - Practice and Experience**, 6 (1976), pp. 61-69.

[15] HATTORI, Mário T. - Uma Disciplina de Programação em FORTRAN-66. DSC/UFPb, Relatório Técnico, Maio/1984.

[16] KROGH, Fred T. - Features for FORTRAN Portability, **Portability of Numerical Software**, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 361-367.

[17] LARMOUTH, J. - FORTRAN 77 Portability, **Software- Practice** & Experience, 11 (1981), pp. 1017-1117.

[18] LOVEMAN, David B. - Program Improvement by Source to Source Transformation, **3rd ACM Symposium on Principles of Programming Languages**, Atlanta, Georgia, Jan/1976, pp. 140-152.

[19] MAHER, B. & Sleedman, D. H. - Automatic Program Improvement: Variable Usage Transformations, **ACM Trans. Program. Lang. Syst.**, 5 (1983), pp. 236-264.

[20] MEDEIROS, Camilo de L. G. - Projeto e Implementação de um Sistema Transformador de Programas FORTRAN, **Dissertação de Mestrado**, UFPb (1986).

[21] MEDEIROS, Camilo de L. G.; Mongiovi, Giuseppe; Hattori, Mário Toyotaro - Sistemas Transformacionais de Programas FORTRAN: Um Estudo de Caso, **Anais do III ERMAC**, Natal-RN (1986).

[22] PARTSCH, H. & Steinbrueggen, R. - Program Transformation Systems, **Computing Surveys**, 15 (1983), pp. 199 - 236.

[23] POOLE, P. C. & Waite, W.M. - **Portability and Adaptability - Software Engineering - An Advanced Course**. F. L. Bauer (ed.), Springer-Verlag, pp. 183-277.

[24] RYDER, B. G. - The PFORT Verifier, **Software- Practice and Experience**, 4 (1974), pp. 359-377.

[25] SCOWEN, R. S. - Some Aids for Program Documentation, **Software - Pratice and Experience**, 7 (1977), pp.779-792.

[26] STANDISH, Tomas A.; Kibler, Dennis F.; Neighbors, James M.- Improving and Refining Programs by Program Manipulation, **Proceedings of ACM Annual Conf** - Houston, Texas, 1976, pp.509-516.

[27] TANENBAUM, A. S.; Klint, P.; Bohm, W. - Guidelines for Software Portability, **Software - Practice and Experience, 8** (1978), pp.681-698.

[28] TASSEL, Dennis Van - **Program Style, Design, Efficiency, Debugging and Testing.** Prentice-Hall, Englewood Cliffs, N.J., 1974.

TIPO DE ENTIDADE	AÇÃO DO SISTEMA
Identificador	Se for possível, converte o nome do identificador de acordo com as "Convenções de Nomenclatura"(veja seção 3.3.1 de [20] da versão anterior para a versão atual.
Constante	Converte a sintaxe das constantes tipo REAL ou COMPLEX, p. ex.,0.5E0 para 0.5D0.
Nome de Precisão	Substitui o nome da precisão anterior pelo nome da precisão atual.
Todos os demais	Nenhuma (O texto fica inalterado)

Tabela I : Ações de Módulo de Conversão da Documentação Interna.

INSTRUMENTADOR DE PROGRAMAS "C"

Marcelo S. Pimenta Verner Heidrich Ana M. A. Price

Universidade Federal do Rio Grande do Sul
Porto Alegre - Brasil

Resumo

Descreve-se um instrumentador para a linguagem "C", que possibilita o uso de comandos especiais visando facilitar a depuracao de programas. A ferramenta foi implementada com o objetivo de proporcionar ao programador um recurso que permita o melhor acompanhamento da execucao de programas, facilitando a localizacao de erros e identificacao de suas causas.

Os comandos de depuracao são especificados como comentários no programa fonte, de modo que, concluída a fase de teste, o programa pode ser imediatamente compilado e colocado em operacao, dispensando a remocao daqueles comandos.

O instrumentador é um pré-processador, que recebendo um programa em "C" instrumentado, analisa declaracoes-fonte e comandos de depuracao, e traduz os últimos em instrucoes "C", entregando ao compilador o programa original com as adicoes para depuracao.

Sao apresentados, tambem, detalhes da implementacao do instrumentador proposto.

1. Introducao

O teste de programas pode absorver acima de 50% do custo de desenvolvimento de sistemas automatizados[Tai 80]. Uma das razoes do alto custo da fase de testes é o caráter imprevisível da depuracao. Depuracao é a atividade que inicia com a descoberta de erros no programa e compreende, basicamente, a localizacao e correcao destes erros. A identificacao de um erro pode demorar minutos, horas e até dias. Existem dados que mostram que a maioria dos erros de programas sao encontrados e corrigidos em aproximadamente uma hora. Entretanto, existe um conjunto de erros (em torno de 20%) onde cada elemento toma da ordem de um dia para ser identificado e corrigido [Johnson 83].

A depuracao de um programa é uma atividade difícil e dispendiosa, particularmente, por dois motivos :

- porque o programador desenvolve mentalmente um modelo de comportamento do programa que nao corresponde exatamente ao seu funcionamento real; e
- pela falta de ferramentas adequadas a este tipo de atividade.

A ferramenta ideal para depuracao deve ser interativa, facil de usar e oferecer facilidades tais como [Ward 86]:

- possibilidade de executar o programa passo-a-passo, mostrando os comandos sendo executados e os valores das variáveis envolvidas;
- suspender a execucao do programa em pontos especificos ou na ocorrencia de eventos pré-determinados;
- permitir ao usuario acesso ao codigo fonte para modificacao de instrucoes e alteracao de dados;
- permitir compilacao incremental e registrar os comandos de depuracao executados de modo a possibilitar a repeticao do processo.

Entretanto, a implementacao de tais facilidades nao constitui um procedimento trivial e pode sofrer sérias restricoes quando desenvolvida para microcomputadores,pois exige espaco de memória apreciável, considerando que devem permanecer em memória varias estruturas geradas durante a compilacao do programa ,juntamente com os interpretadores para o programa fonte e para os comandos de depuracao requeridos pelo usuario.

Uma tecnica de depuracao mais acessivel é a instrumentacao de programas fonte antes da compilacao. A instrumentacao de programa permite determinar certas propriedades (por exemplo, identificacao de caminhos inacessiveis, frequencia de execucao de instrucoes, anomalias no fluxo de dados, etc) que possivelmente só seriam identificadas a um custo elevado por outras técnicas de verificacao.

As formas mais frequentes de instrumentacao de programas incluem :

- insercao de contadores para medir niveis de profundidade de teste;
- adicao de assercoes para verificar a validade de valores de variaveis; e
- exibicao de valores de variaveis em determinados pontos do programa.

Entretanto, muitas das funcoes pertinentes a depuradores simbolicos sao passiveis de serem implementadas atraves de instrumentacao, como por exemplo execucao passo-a-passo, exibicao dos comandos sendo executados, suspensao da execucao do programa em pontos predeterminados e entrada de valores para variáveis durante a execucao do programa.

A maioria das implementacoes de linguagens existentes no mercado nao possui ambiente de depuracao simbolica. Fica a cargo dos proprios programadores o desenvolvimento de procedimentos de depuracao, que acabam desviando sua atencao do problema a que se propunham resolver. Um ambiente desejável seria aquele que tornaria os detalhes de depuracao transparentes ao programador. A necessidade de uma ferramenta de depuracao com tal requisito é que deu origem a este trabalho.

O protótipo descrito neste artigo resultou de um trabalho proposto na Disciplina de Compiladores ministrada no Bacharelado em Ciencias da Computacao da UFRGS. Os projetos propostos pela Disciplina visam a implementacao de processadores que envolvam tecnicas de compilacao. Os estudantes têm aproximadamente tres meses para a realizacao do trabalho, sendo este periodo partilhado com o desenvolvimento de projetos propostos em outras disciplinas. O prototipo em questao foi desenvolvido e testado, em tempo hábil, por dois dos autores deste relatório.

O assunto do trabalho foi escolhido por condizer com o tema apresentado na Disciplina, o prazo cabível de desenvolvimento e por vir ao encontro da comunidade usuária local, como instrumento de aplicacao prática imediata. Os implementadores pretendem extende-lo com novas funcoes e aplicá-lo no desenvolvimento de programas na indústria onde trabalham.

A linguagem "C" foi escolhida como linguagem hospedeira devido a :

- suas características de portabilidade, o que faz com que o instrumentador possa migrar para vários equipamentos, e
- expressiva comunidade de usuarios (pesquisadores, estudantes) atuantes na Universidade.

2. Descricao dos Comandos de Depuracao

Os comandos de depuracao, aqui descritos, foram projetados para obterem informacoes sobre o programa, através de uma linguagem simples e facilmente memorizável pelo programador.

A fim de que o programa "C" instrumentado nao precise ser modificado para ser submetido ao compilador, os comandos de depuracao devem ser escritos na forma de comentarios, isto é, encerrados entre os pares '/*' e '*/'. Assim, qualquer erro na escrita ou formulacao dos comandos de depuracao fará com que sejam interpretados como comentários.

A sintaxe, a semantica e a interpretacao dos comandos é apresentada a seguir.

⟨comando_inicio_depuracao⟩ ::= /* DEPURAR */

O comando DEPURAR delimita o início do segmento de programa a ser depurado. Se tal comando não estiver presente no programa, os comandos de depuracao posteriores serao desprezados. Dois ou mais comandos DEPURAR consecutivos têm o valor de um único.

⟨comando_exibicao⟩ ::= /* EXIBIR ⟨lista_de_variaveis⟩ */

Mostra, no dispositivo de exibicao,o valor corrente das variaveis indicadas em ⟨lista_de_variaveis⟩, no formato "⟨var⟩ = ⟨valor⟩".

As variaveis podem ser quaisquer variaveis validas em "C", que tenham sido declaradas previamente. Se nao forem reconhecidas é gerada, a tempo de pré-processamento, uma mensagem de alerta.

⟨comando_assercao⟩ ::= /* VERIFICAR ⟨expressao⟩ */

Realiza o teste da assercao indicada quando da execucao daquele segmento do programa. Se a assercao for violada, isto é, se a expressao avaliada resultar em FALSO, será exibida a mensagem "ASSERCAO ⟨expressao⟩ VIOLADA".

⟨expressao⟩ deve ser uma expressao valida na linguagem "C", impondo-se as seguintes restricoes :

(i) expressoes nao podem conter variaveis do tipo 'structure', 'union' e 'typedef',

(2) expressoes nao devem modificar o valor das variaveis envolvidas, por exemplo :

++x, x= a--, --x,

(3) expressoes nao podem conter : conversao de tipo, a funcao pré-definida 'sizeof' e expressoes condicionais envolvendo o operador ternario '?'.

(4) lista de expressoes nao sao permitidas, por exemplo :

x !! 4, w & 0x2f.

Expressoes do tipo (2) nao podem aparecer em comandos de depuracao, pois quando avaliadas, modificam o valor das variaveis envolvidas. Tais expressoes serao avaliadas se o instrumentador for ativado para gerar o codigo correspondente ao comando VERIFICAR. Caso contrario, elas serao vistas como comentarios. Desta maneira, se o programa for instrumentado, as variaveis envolvidas terao um valor, senao, terao outro. O que deseja-se evitar com a restricao é o reflexo indesejado da depuracao na execucao do programa. As demais restricoes apenas reduzem a dificuldade de implementacao do instrumentador.

A opcao de aceitar a mesma sintaxe das expressoes da linguagem "C" em <expressao> na assercao facilita ao programador, pois ele nao necessita memorizar novos formatos, operadores e regras de sintaxe.

<comando_rastreamento> ::= /* RASTREAR */

O comando RASTREAR delimita inicio do segmento de programa que será rastreado. Rastrear significa exibir a imagem do comando <sequencial> sendo executado. Isto é, a partir do ponto marcado pelo <comando_rastreamento> todo Comando Significativo executado é mostrado no dispositivo de exibicao. Comando Significativo é todo aquele que nao é Comando de Controle de Fluxo de Programa (CCFP). Em "C", os CCFP sao : 'if', 'if - else', 'switch - case - default', 'while', 'do - while' e 'for'. Os comandos que compoem os campos dos CCFP sao Comandos Significativos se nao forem CCFP.

Dado o segmento de programa abaixo, somente os comandos especificados nas linhas 3, 4, 6, 8 e 10 sao passiveis de exibicao quando rastreados :

```
1.   while (x) {
2.         if (w) {
3.              c++;
4.              break;
5.              }
6.         a = proc(x,c);
8.         b = a >> w & 0x80;
9.         }
10.  return (a);
```

<comando_fim_rastreamento> ::= /* NAO RASTREAR */

O comando NAO RASTREAR delimita o fim do segmento de programa que estava sendo rastreado. Se omitido, é assumido como fim do segmento o final do programa. Entre pares de comandos de rastreamento, podem ser especificados os comandos EXIBIR, VERIFICAR, PARAR E PASSO A PASSO.

<comando_parada> ::= /* PARAR */

O comando PARAR define um ponto de parada ("breakpoint") no programa. No momento de sua execucao, é solicitado do programador uma diretiva dentre as possíveis : continuar execucao sequencial ou encerrar o programa. Isto é feito quando da emissao da mensagem "PARADA : Continuar / Encerrar (C/E) ? _".

<comando_inicio_passo_a_passo> ::= /* PASSO A PASSO */

O comando PASSO A PASSO delimita o início do segmento de programa no qual a execucao de cada Comando Significativo é condicionada à intervencao do programador, através de mensagens semelhantes às do comando PARAR.

<comando_fim_passo_a_passo> ::= /* FIM PASSO A PASSO */

O comando FIM PASSO A PASSO delimita o final do segmento iniciado com PASSO A PASSO. Se omitido, é assumido o final do programa.

<comando_fim_depuracao> ::= /* NAO DEPURAR */

O comando NAO DEPURAR delimita o final do segmento de depuracao. A partir do ponto demarcado por este comando, qualquer comando de depuracao posterior será considerado comentário. Se omitido, é assumido o final do programa.

Podem existir vários pares de comandos de início e fim de depuracao num mesmo programa fonte.

3. Implementacao

O instrumentador foi implementado como um pré-processador que analisa codigo fonte "C" contendo comandos de depuracao e gera um programa "C" instrumentado. Este constitui-se em um programa com modificacoes e adicoes feitas pelo pré-processador para implementar as facilidades dos comandos de depuracao.

Esta opcao nos pareceu mais imediata pelos seguintes motivos : a implementacao era viável no prazo estabelecido, considerando-se ser um trabalho de cunho academico; os comandos poderiam ser implementados independentemente um dos outros, de modo que se poderia implementar uma versao básica e, depois, incrementá-la com novos comandos. Além disso, se implementado desta forma, o instrumentador se tornaria facilmente adaptável a outras linguagens de programacao, bem como outros microcomputadores devido à portabilidade da linguagem "C" e ao método tabular de análise implementado pelo instrumentador.

O pré-processador implementado foi desenvolvido em linguagem "C", sob o Sistema Operacional MS-DOS em microcomputador PC-compatível nacional. "C" foi a linguagem escolhida devido a suas características de modularidade, portabilidade e sua expressiva utilizacao pela comunidade academica local no desenvolvimento de software de suporte.

3.1. Estrutura do Sistema

O pré-processador é constituído pelos seguintes módulos
- Analisador Léxico,
- Analisador Sintático e
- Gerador de Comandos,

os quais são descritos a seguir.

Analisador Léxico

O Analisador Léxico reconhece basicamente seis tipos de tokens: identificadores, palavra-chave, constantes, operadores, separadores e cadeias de caracteres. Interage com o Analisador Sintático, cada vez que esse o solicitar, devolvendo o próximo 'token' válido ou uma indicacao de erro.

Análisador Sintático

O Analisador Sintático trabalha em duas etapas :
- Pré-análise, que identifica as figuras sintáticas chave (que interessam à depuracao) e só entao desencadeia o processo de análise sintática. Evita-se assim a análise completa do código fonte, que é desnecessária segundo os objetivos do instrumentador.
- Análise, implementada através de um analisador 'top-down' nao-recursivo, baseado na técnica do grafo sintático de Wirth com funcionamento determinístico [Setzer 83], adequado aos objetivos do instrumentador.

Gerador de Comandos

O Gerador de Comandos tem por funcao acrescer ao programa fonte trechos de código que implementam as tarefas de depuracao especificadas pelo programador. O Gerador realiza sua funcao a partir de esqueletos de comandos, que são completados de acordo com as características específicas de cada programa (nomes de variáveis, funcoes declaradas e outros).

3.2. Funcionamento do Instrumentador

Uma linha do fonte é lida a cada vez, sendo percorrida pelo Analisador Léxico, que a separa em elementos sintáticos ('tokens'). Como apenas os elementos, declaracao de variáveis e comandos de depuracao do programa fonte, precisam ser analisados para a depuracao, deve haver um processo de identificacao destes elementos antes da análise, o qual é feito na fase de pré-análise do Analisador Sintático.

Sao identificadas apenas as figuras sintáticas chaves de estruturas que interessam à depuracao para entao desencadear o processo de análise sintática de cada estrutura. Se a estrutura é significativa para a depuracao, o Analisador Sintático é acionado.

Quando uma declaracao de variável, ou de funcao, é reconhecida, sao feitas inclusoes e alteracoes na tabela de símbolos.

Quando é reconhecido um comando de depuracao, o gerador de comandos é chamado.

Por fim, as linhas lidas, alteradas ou nao pelo gerador de comandos sao gravadas no arquivo destino da instrumentacao, gerando o programa instrumentado que poderá ser submetido a um processo compilativo normal. Na figura 1 é apresentado o diagrama funcional do sistema.

3.3. Dispositivo de Exibicao

O programador pode especificar qual o dispositivo que pretende utilizar para a exibicao dos resultados da depuracao. Um dispositivo pode ser um arquivo do tipo texto ou um arquivo especial do sistema operacional, como por exemplo, a impressora ou o vídeo. Isto é indicado na chamada do instrumentador, passando-se como parâmetro o nome do arquivo a conter os resultados da depuracao.

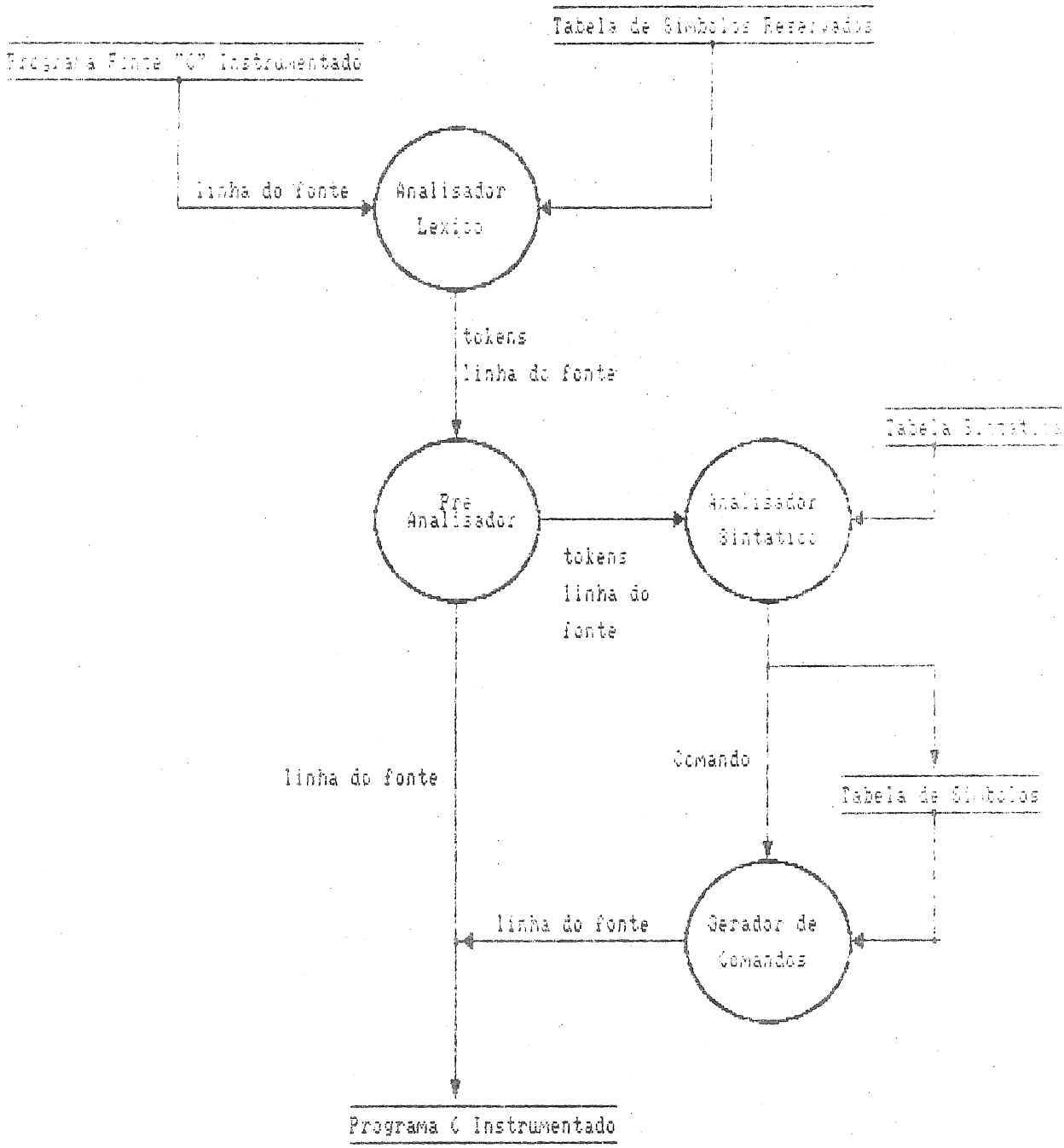


Figura 1. Diagrama Funcional do Sistema.

De forma geral, a execucao do instrumentador é efetuada como segue :

- para disparar a instrumentacao sobre o programa gerarq.c, redirecionar o dispositivo das mensagens de instrumentacao para o arquivo depura.err e gerar os futuros resultados da depuracao no arquivo depura.sai :

```
A> depura -igera_arq.c -ogera_arq.dep -edepura.sai > depura.err
```

- para verificar possíveis mensagens de erros da instrumentacao :

```
A> type depura.err
```

- para compilar o programa instrumentado :

```
A> compila gera_arq.dep
```

- para executar o programa :

```
A> gera_arq
```

- para verificar o resultado da depuracao :

```
A> type depura.sai
```

4. Conclusoes

Instrumentacao de programas é um meio economico de obter informacoes adicionais sobre a execucao de um programa como um produto suplementar do teste de programas. É razoavelmente simples de ser implementada e fácil de ser usada por programadores pois, ao contrario de outras tecnicas de verificacao e validacao de programas, nao requer conhecimento profundo de matematica ou logica formal [Huang 80].

Foi apresentada a descricao de um instrumentador para programas em "C" visando facilidades para depuracao. Este trabalho foi desenvolvido com o objetivo de propiciar ao programador comandos especiais de depuracao, de fácil aprendizagem, que permitam obter informacoes sobre a execucao de seus programas.

Os comandos de depuracao sao especificados como comentários, de modo que, finda a fase de depuracao, o programa em teste é compilado e liberado para operacao sem exigir quaisquer alteracoes.

O instrumentador foi implementado como um pré-processador que analisa deçlaracoes de dados , funcoes do programa fonte e comandos de depuracao entre delimitadores de comentários, e traduz estes últimos em codigo fonte "C".

Os comandos de depuracao implementados incluem marcadores de início e fim de depuracao, exibicao de valores de variáveis, verificacao de assercoes, delimitadores de início e fim de execucao passo-a-passo, e parada de programa. A implementacao, tal como foi desenvolvida, permite, com razoável facilidade, a insercao de novos comandos de depuracao e, também, a adaptacao do instrumentador a outras linguagens de programacao (isto deve-se ao fato de que a análise sintática foi implementada através de um método tabular).

Este trabalho resultou de um projeto proposto na Disciplina de Compiladores do Bacharelado em Ciências de Computacao da UFRGS. Como programadores experientes em "C", os implementadores acreditam na praticidade da ferramenta proposta.

Referências Bibliográficas.

A. Aho & J. Ullman, Principles of Compiler Design, Adison Wesley, 1977.

D. Gries, Compiler Construction for Digital Computers. John Wiley, New York, 1971.

[Huang 80]
J. C. Huang, Program Instrumentation : A Tool for Software Testing : 2, in INFOTECH State of The Art Report, U.K., 1979.

[Johnson 83]
M.S. Johnson(ed.),Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on High-Level Debugging,
ACM Software Engineering Notes,Vol.8,No.4,Aug 1983.

B. Kernighan & D. Ritchie, The C Programming Language, Prentice-Hall, 1978.

[Setzer 83]
Setzer, Valdemar W., A Construcao de um Compilador, Editora Campus, 1983.

L. G. Stucki & G. L. Foshee, New Assertion Concepts for Selfmetric Software Validation, Proc 1975 International Conference on Reliable Software (April 75) in SIGPLAN Notices Vol.10,No.6,Jun 1975.

[Tai 80]
K. Tai, Program Test Complexity and Test Criteria, IEEE Transactions on Software Engineering, Vol.6, No. 6, 1980.

[Ward 86]
R. Ward, Debugging C, Indianapolis, QUE Corporation, c 1986.

R. T. Yeh, Current Trends im Program Methodology vol II: Program Validation, Prentice Hall, Englewoods Cliffs, NJ , 1977.

Saltos no Locales: Una extensiòn a Turbo-PASCAL

Rodrigo Lòpez B.
Universidad de los Andes
Facultad de Ingenieria
Departamento de Sistemas y Computaciòn
Apartado Aèreo 4976
Bogotà - Colombia

RESUMEN

En los ambientes de programaciòn interactivos son tipicos los mòdulos que detectan y se recuperan de los errores; en consecuencia, el flujo normal del control dentro de los programas se altera de manera impredecible al detectar esas situaciones anormales. La justificaciòn clàsica de la presencia de los saltos incondicionales (GOTO) en Pascal ha estado relacionada con una soluciòn "estructurada" para el tratamiento de errores; sin embargo, en Turbo-PASCAL el alcance de los saltos incondicionales es local al procedimiento en donde se definen las etiquetas (label) lo que impide efectuar saltos desde el interior de un procedimiento hacia el final de un programa (por ejemplo).

En este articulo se muestra la implementaciòn de una estrategia de soluciòn para el tratamiento de errores en Turbo_PASCAL, similar a las construcciones ERRSET-ERROR o CATCH-THROW de Lisp. La soluciòn es muy eficiente pues consta de unas pocas lineas en lenguaje ensamblador y es muy sòlida pues las verificaciones que atañen al flujo del control se realizan ùnicamente en dos puntos del programa.

I INTRODUCCION

A pesar de las múltiples críticas que ha recibido la primitiva de control GOTO, a través de la historia de la programación, parece haber consenso (aún entre sus detractores) acerca de su utilidad en situaciones muy precisas, en donde son válidos tanto los argumentos de eficiencia en la ejecución como los de claridad en la programación. De hecho, en lenguajes "estructurados" como PASCAL, se incluyó el GOTO en la versión original aun cuando se hacen advertencias con respecto a su uso [JEN75].

En la versión más popular de PASCAL para microcomputadores de 8 y 16 bits, Turbo-PASCAL [BOR84], existe el GOTO pero su alcance es local. En este articulo se discute una extensión a Turbo-PASCAL (en su versión para sistema operacional MS-DOS) por medio de la cual se pueden efectuar saltos no-locales dentro de un programa. Se ilustra la necesidad de tal facilidad con situaciones tomadas de la implementación de un ambiente de programación real que se utiliza actualmente en el primer curso de programación en la Universidad de los Andes [LOP87a], [LOP87b].

II EL SISTEMA Turbo-KAREL.

El sistema Turbo-KAREL es un ambiente de programación autocontenido que se utiliza como introducción a la programación en un lenguaje imperativo (tipicamente PASCAL). El lenguaje de programación utilizado es KAREL y la implementación realizada en la universidad de los Andes se ajusta a la definición hecha por Richard Pattis. KAREL es un robot que se desplaza en un "mundo" cartesiano en donde puede haber obstáculos (muros) y un tipo particular de señales (pitos); para más aclaraciones se puede consultar [PAT81]. Este proyecto se adelantó con financiación del Fondo Nacional de Proyectos de Desarrollo del gobierno colombiano, funciona en microcomputadores del tipo PC-IBM o compatibles y es de distribución gratuita dentro de Colombia.

La implementación del sistema se realizó enteramente en Turbo-PASCAL y consta, además de un "compilador" y un intérprete para los programas de karel, de un editor de programas y de un editor gráfico e interactivo para los mundos . Editar un mundo consiste en definir los muros, los pitos y la posición inicial del robot. Un programa "compilado" se "interpreta" sobre un mundo seleccionado.

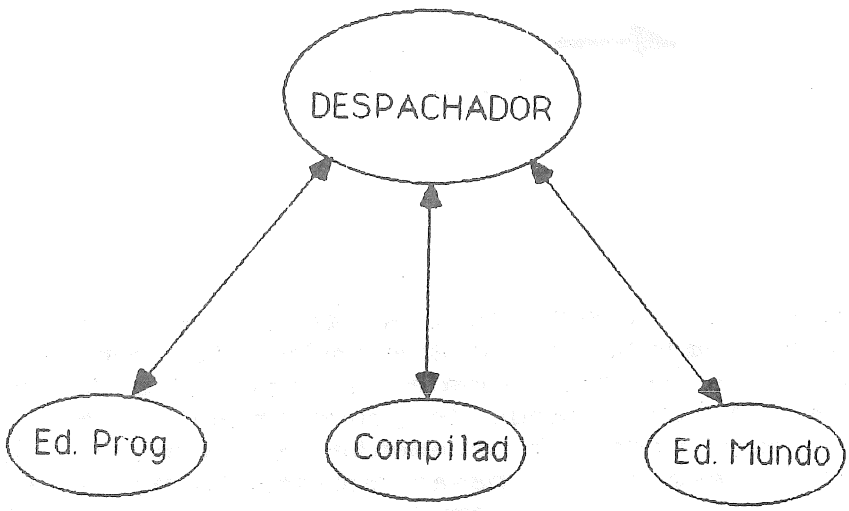
III. FLUJO NORMAL DEL CONTROL EN Turbo-KAREL

La estructura de control de Turbo-KAREL es similar a la de muchos sistemas interactivos en los que un despachador de comandos distribuye el control a diferentes módulos servidores y éstos lo

retornan, a su turno, al despachador de comandos [TOR87]. Los mòdulos servidores en Turbo-KAREL son:

- Editor de Programas
- Editor de Mundos
- Compilador
- Intèrprete.
- Otros: Conexiones con las facilidades del sistema operacional residente.

En el modo normal de operaciòn el control fluye ùnicamente en las direcciones especificadas por las flechas del siguiente diagrama.

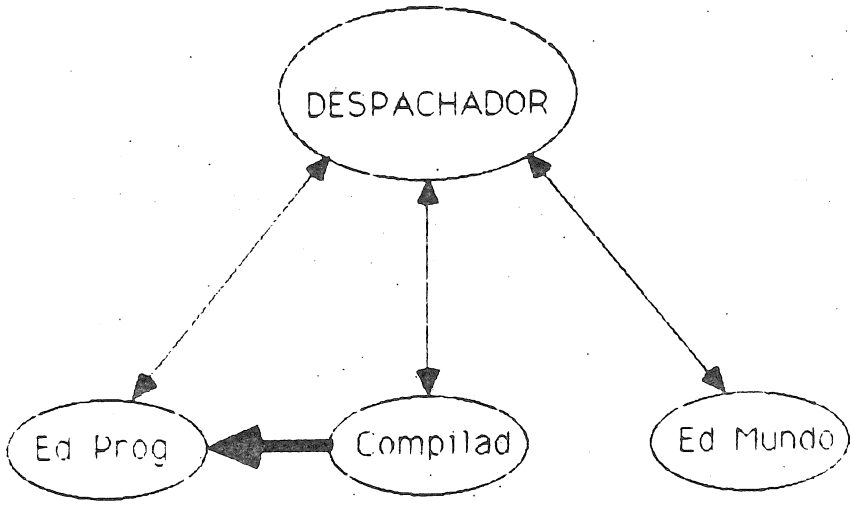


IV FLUJO ANORMAL DEL CONTROL: Recuperaciòn de Errores.

En Turbo-KAREL se manejan los errores tipicos que comete el usuario de un ambiente de programaciòn:

- Errores de sintaxis, manejados por el compilador.
- Errores de ejecuciòn, manejados por el intèrprete.

En los dos casos se suspende la ejecuciòn del servicio solicitado (compilaciòn o interpretaciòn) y el usuario puede elegir, retornar al despachador de comandos (flujo normal del control) o pasar directamente a editar el programa en el sitio en donde sobrevino el problema (flujo anormal del control).



En un sistema de las características anteriores, programado en PASCAL, presumiblemente el despachador de comandos es un procedimiento y la manera de ceder el control a los módulos de servicio es la clásica llamada de procedimiento. No es posible con este esquema pasar bruscamente el control entre dos servicios: es imprescindible pasar nuevamente por el despachador de comandos. En el caso, por ejemplo, de un error de compilación, la dificultad no estriba en "saber cuál fué la línea del texto en donde apareció el error" sino en lograr que desde un sitio "totalmente arbitrario" en la ejecución, pueda cedérsele el control a un procedimiento específico, "olvidándose de casi todo lo que había ocurrido".

Ilustremos con un esquema de programa en Pascal, cuál es la situación. Consideremos el siguiente programa:

```
Program Ejemplo;
  Procedure P;
    Procedure Q;
      Procedure R;
        Procedure S;
         Begin
         End;
        Begin
        End (* de R *);
      Procedure T;
        Begin
        End (* de T *);
      Begin
      End (* de Q *);
```

```
    Procedure U;
       Begin
       End (* de U *);
    Begin
    End (* de P *);
 Begin
 End. (* de Ejemplo *)
```

Según las reglas de alcance de Pascal, la siguiente sucesión de llamadas es válida:

Ejemplo llama a P llama a Q llama a Q llama a Q llama a T llama a T llama a R llama a S llama a S llama a S

Supongamos que en este instante de la ejecución se quiera ceder el control al procedimiento U. Dadas las restricciones del alcance, la solución sería:

- Retornar bruscamente el control a P (con todo lo que ello implica para los ambientes de variables locales y parámetros de los procedimientos activos entre P y S).

- Que P le ceda el control a U.

Recordemos que en TurboPascal, todos los GOTOs son locales al procedimiento en donde se definen, lo que hace imposible forzar un retorno directo a P con el fin de que él se encargue de la redistribución del control. Una solución sencilla, pero extremadamente frágil, en TurboPascal, sería utilizar una variable global que es colocada en True cuando aparece la situación que requiere de ese retorno especial. Entonces, el valor de esa variable debería ser consultado por TODOS LOS PROCEDIMIENTOS POTENCIALMENTE ACTIVOS para que ellos retornen el control "a quien los invocó" y de esta manera, en algún momento se llega a P quien toma las acciones apropiadas. Es evidente que esta solución, aun cuando puede funcionar, exige demasiado código adicional y no es sencillo certificar su validez. Por otro lado es muy ineficiente pues requiere que todos los procedimientos involucrados, verifiquen explícitamente la ocurrencia o ausencia de la situación anormal para tomar las providencias adecuadas.

V UNA SOLUCION CLASICA IMPLEMENTADA EN Turbo-PASCAL

En la situación descrita anteriormente, cuál es el problema de fondo?. Es necesario hacer retornar todos los procedimientos activos ya que los ambientes de datos de cada uno de ellos (Parámetros y variables locales) se convertirían en objetos carentes de sentido si el control fuera bruscamente a P. Dado que estos ambientes se manejan en una pila, una solución alterna

sería:

- Marcar el ambiente activo en el sitio que va a retomar el control (Dentro del procedimiento P).
- Marcar la dirección de retorno en ese mismo punto.

Estas dos marcas se conocen como el "contexto de ejecución" en un instante dado [PRA81], ya que ellas definen la próxima instrucción a ejecutarse junto con los datos que tienen sentido en ese momento. Finalmente, el último paso sería:

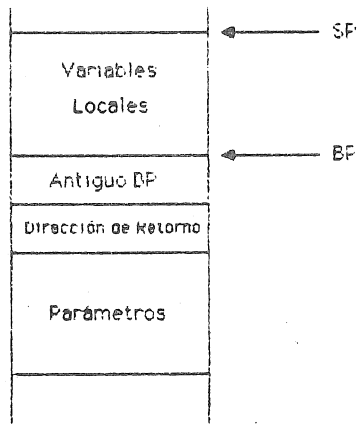
- Utilizar las marcas (recuperar el contexto), en el sitio en donde se quiera forzar un retorno brusco.

En nuestro ejemplo, el contexto se marca en dos sitios diferentes:

- Dentro de P se marca el nivel de la pila de ambientes.
- Dentro de Q se marca la dirección de retorno a P!!!. Esto es importante pues es el procedimiento Q quien conoce el punto de continuación del control dentro de P, ya sea que Q retorne normalmente o que se produzca un retorno brusco a partir de algún procedimiento invocado directa o indirectamente por Q.

y se recupera en S cuando así se requiera.

Para lograr todo lo anterior en turbo pascal, es preciso manipular a bajo nivel la pila de ambientes puesto que (afortunadamente) no es posible hacerlo directamente con instrucciones de Pascal. Según se explica en el manual de TurboPascal, un ambiente local está definido por los valores de los registros BP y SP del procesador 8088 (ver figura).



Las variables locales se encuentran encerradas entre estos dos registros, cuyos valores son direcciones tales que

ContenidoDe(BP) >= ContenidoDe(SP).

Por otro lado, desde la direcciòn apuntada por BP, hacia las direcciones superiores en la memoria, el contenido es:

- Antiguo valor de BP.
- Direcciòn de retorno del procedimiento activo. Note que este valor corresponde a la direcciòn de una de las instrucciones del procedimiento que invocò al que està activo en este momento.
- Parâmetros del procedimiento activo.

Con esta informaciòn, unas pocas lineas en assembler resuelven el problema. Es recomendable utilizar variables enteras y globales para almacenar los valores de los registros. El còdigo de màquina se inserta por medio de la instrucciòn InLine de TurboPascal en los sitios adecuados, con la ventaja de poder mencionar "por su nombre" las variables globales requeridas. Concretamente, supongamos las siguientes declaraciones globales:

```
Var
   Base:Integer;       (* Valor de BP *)
   Tope:Integer;       (* Valor de SP *)
   Retorno: Integer;   (* Direcciòn de Retorno *)
```

Los trozos de còdigo serian:

- Marcar el ambiente de datos:

```
Mov ax,bp
Mov Base,ax
Mov ax,sp
Mov Tope,ax
```

- Marcar la direcciòn de retorno:

```
Mov ax,[bp+2]  (* Pues el antiguo BP ocupa 2 bytes *)
Mov Retorno,ax
```

- Recuperar el contexto:

```
Mov ax,Tope
Mov sp,ax
Mov ax,Base
Mov bp,ax
Mov ax,Retorno  (* Hasta aquí se recuperò todo el
                   contexto *)
```

```
Push ax        (* Coloque la dir de retorno en la pila *)
Ret            (* Retorne *)
```

En nuestro sistema Turbo-KAREL se adoptò esta soluciòn, la que por otro lado es muy similar a las construcciones de alto nivel ERRSET-ERROR o CATCH-THROW de algunos dialectos de Lisp. Concretamente, en el manejo de errores de compilaciòn se requiere:

- El despachador de comandos marca el ambiente de datos antes de cederle el control al compilador.
- La llamada al compilador se hace a travès de un procedimiento "fantasma" cuya ùnica funciòn es marcar el punto de retorno en el despachador e invocar al verdadero compilador.
- El anàlisis sintàctico se realiza con el mètodo del descenso recursivo lo que lleva a situaciones como las descritas anteriormente. Si ocurre un error, quien lo detecte, llama al procedimiento ERROR quien es el encargado de recuperar el contexto para el retorno brusco al despachador de comandos.
- Al recuperar el control, el despachador de comandos verifica el valor de una variable que le indica si el retorno fue normal o anormal. Esta variable es manipulada ùnicamente por el procedimiento ERROR.

Como se ve, la soluciòn es muy sòlida pues solamente se tiene conciencia de la situaciòn anormal en dos puntos "extremos" del programa: en el despachador de comandos y en el procedimiento de ERROR. Los demàs procedimientos involucrados directa o indirectamente, reciben, ceden y retornan el control sin ocuparse de los problemas potenciales. Finalmente, es muy eficiente pues el còdigo requerido se reduce estrictamente a las 13 instrucciones en lenguaje ensamblador (mostradas anteriormente) màs la escasa manipulaciòn de la variable de "comunicaciòn" conocida por el despachador de comandos y el procedimiento de ERROR.

Por otro lado, trae consecuencias en cuanto al "estilo de programaciòn" en los sitios en donde se detectan los errores, en donde aparece claramente que el control interrumpe su flujo normal. El siguiente es un trozo de programa, tomado de uno de los procedimientos del analizador sintàctico, que ilustran este estilo inhabitual (pero estructurado) en Pascal:

```
Case SiguienteToken of
   TokenIterate:
      Begin
        p:= ConsNi(SiguienteToken,Nil,Nil,Linea);
        DemeSiguienteToken;
        If (SiguienteToken <> Numerico) then ERROR(19);
        If (ValorToken > 255) then ERROR(21);
        q:=ConsNi(ValorToken,Nil,Nil,Linea);
        p^.Cuerpo := q;
        If (SiguienteToken <> TokenTime) then ERROR(20);
        DemeSiguienteToken;
        q^.Cuerpo:= AnaliceInstruccion;
      End;
 .......       etc    .......
```

Obsèrvese que, por la forma del programa, es evidente que el procedimiento ERROR no retornarà el control a la siguiente instrucciòn.

VI CONCLUSIONES

Con este ejercicio de programaciòn hemos corroborado que Turbo-Pascal se adapta a casi cualquier necesidad de programaciòn, de manera clara y eficiente. La situaciòn esbozada en el articulo es tipica de los sistemas que detectan y se recuperan de los errores; nuestra soluciòn aparece como un "esquema de control" que puede adoptarse fàcilmente en Turbo-PASCAL, es muy eficiente, confiable e introduce en PASCAL un estilo estructurado para saltos no-locales, similar a las construcciones ERRSET-ERROR o CATCH-THROW de algunas versiones de Lisp.

REFERENCIAS

[BOR86] Borland Inc.
"Turbo-PASCAL: Turbo Tutor"
Borland International Inc, Scotts Valley, U.S.A., 1984.

[JEN75] Jensen Kathleen, Wirth Niklaus.
"PASCAL User Manual and Report".
Springer-Verlag, New York, U.S.A., 1975.

[LOP87a] Lòpez Rodrigo, Toro Victor.
"Turbo-KAREL: Manual del Sistema"
Publicaciòn CIFI-Uniandes, Bogotà, Colombia, 1987.

[LOP87b] Lòpez Rodrigo, Toro Victor.
"Turbo-KAREL: Libro del Proyecto"

Publicación CIFI-Uniandes, Bogotá, Colombia, 1987.

[PAT81] Pattis Richard.
"Karel the Robot: A Gentle Introduction to the Art of Programming"
Prentice-Hall Int, Englewood Cliffs, U.S.A., 1981.

[PRA81] Pratt Terrence.
"Programming Languages: Design and Implementation"
Prentice-Hall, Eglewood Cliffs, U.S.A., 1984.

[TOR87] Toro Victor.
"Diseño, Especificación y Prototipificación de Sistemas Interactivos".
En este volumen.

A PROBLEMÁTICA DO DESENVOLVIMENTO DE SOFTWARE NUMÉRICO

Tiarajú Asmuz Diverio
Instituto de Informática PUC RS
Cx. P.1429 90620 Porto Alegre RS

RESUMO

Neste trabalho é caracterizado o software numérico (matemático), considerando-se os níveis de atividade no desenvolvimento do software e os aspectos básicos no processo de desenvolvimento do software, bem como características desejáveis em software numérico. Serão caracterizados as filosofias de software Aplicativo e instrucional, com estilo conversacional. É apresentado o SINAI-16, um software Interativo Numérico Aplicativo e Instrucional em micros de 16 bits voltado ao ensino de métodos numéricos de resolução de equação.

INTRODUÇÃO

Bibliotecas de software matemático podem ser vistas como elo de ligação entre os usuários do computador e seus problemas. Mas devido: ao recente desencadear da automação da sociedade; à falta de recursos aplicativos e instrucionais disponíveis no mercado, compatíveis com os microcomputadores existentes no Brasil; à evolução da importância e função de uma documentação consistente e simplificada e às diferentes finalidades comerciais (e não científicas) a que se destinam os softwares; os softwares matemáticos não tem acompanhado o desenvolvimento dos demais setores da Ciência da Computação e de setores que dela se utilizam.

Recentemente, com a utilização do computador no ensino e com o desenvolvimento da pesquisa em todas as áreas do conhecimento, tem-se verificado a necessidade de bibliotecas de softwares matemáticos numéricos, que contenham uma variedade de subrotinas, uma documentação consistente e simples, capaz de solucionar a grande parte das dúvidas do usuário imediatamente e uma exatidão computacional incluindo exatidão nos resultados e tempo de processamento. É preciso, ainda, que se tenha um estilo conversacional, possibilitando uma melhor ligação, interação entre o usuário e a máquina.

Para a pesquisa, existem as bibliotecas como a IMSL (International Mathematical and Statistical Libraries), a NATS (National Activity to Test Software) e a NAG (Numerical Algorithms Group) que são muito usadas. Por outro lado, deve-se iniciar a desenvolver softwares, para se adquirir experiência para um dia se poder desenvolver um produto, o qual seja comercializável. Para isto, se deve conhecer o processo de desenvolvimento, as dificuldades e as características desejáveis em softwares numéricos.

Tem-se que ter a consciência que, nesta etapa inicial, o software a ser desenvolvido dificilmente será absorvido pela indústria, por não se ter tradição em desenvolvimento e pelo fato de que os industriais preferem buscar softwares no exterior, pelas suas qualidades e confiabilidade. Por outro lado, o software poderia ser utilizado no ensino superior, em Universidades, onde, em geral, os recursos são escassos.

SOFTWARE NUMÉRICO - NATUREZA E DIFICULDADES

O termo "software numérico" ou "software matemático", inicialmente, referia-se a programas com uma performance basicamente matemática na computação de problemas referentes a métodos numéricos que resolvem problemas na Ciência e Engenharia, como é descrito em [CR077].

Software numérico é muito mais do que uma rotina programada de tarefa ou um produto da análise numérica, a qual algumas vezes, situa-se entre a matemática e a ciência da computação, sendo tão aplicada para alguns e irrelevante para outros.

Primariamente, o software numérico necessita tolerar erros. A palavra "erro" é inoportuna, porque erros numéricos não são erros no sentido usual da palavra, mas diferenças entre aproximações, que podem ser computadas num período finito de tempo.

Software numérico tem dois princípios característicos: o de tratar com aproximações de números reais e o fato de o intervalo de representação destes números varia de acordo com o computador.

O fato de que computadores diferem no tratamento de números em ponto-flutuante, significa que se precisa considerar classes de computadores. Há algumas décadas atrás, tratava-se do ambiente de um particular computador, mas hoje, considera-se sobre ambientes em classes de computadores, nas classes de linguagens. Estas classes de computadores têm diferentes intervalos numéricos, diferentes precisões e diferentes propriedades de arredondamento. Software numérico é escrito para se trabalhar nestes vários ambientes. Ele necessita ser sensível à precisão do computador, para que não se tente calcular maior exatidão do que é possível em uma máquina particular. Ele precisa ser sensível ao intervalo numérico para não permitir overflows e underflows desnecessários, e estar alerta a particularidades de erros de arredondamento.

Software numérico pode ser particularmente difícil de ser projetado, porque mesmo antes da precisão, intervalo e problemas de arredondamento, muitas tarefas numéricas são, em um sentido, insolúveis. Isto não quer dizer que o problema seja teoricamente impossível. Ele pode ser "resolvido", podendo ter como "soluçãO":

a) Um resultado correto e com exatidão requerida ou esperada pelo usuário;
b) Uma parte do programa dá que a tarefa é impossível, por não convergência ou ocorrência de algum erro;
c) Uma solução sem a exatidão desejada.

Os tipos de problemas numéricos podem ser classificados, de acordo com C. W. Gear em [GEA80], segundo três critérios que são as propriedades dos dados, propriedades dos algorítmos e propriedades do erro de arredondamento. A análise e descrição destes tipos de problemas vão além do propósito deste trabalho, embora se constate a dificuldades do projetar softwares numéricos.

Gear, ainda, menciona que grande parte das dificuldades do projeto de software numérico é devido à perda da base científica em ambiente computacional. Não existindo critérios simples que possam ser otimizados, mas um conjunto mal-definido de objetos como a confiabilidade, generalidade e utilidade que se está "tão longe" de falhar em quantificar.

Ele considera também o software numérico como parte da ciência da computação e espera que seja estudado em departamentos de ciência da computação ou informática.

NÍVEIS DE ATIVIDADES NO DESENVOLVIMENTO DE SOFTWARE

Existem alguns níveis distintos de atividades no desenvolvimento de software numérico. O trabalho feito em cada nível particular é um pouco independente do trabalho feito em outro nível, embora as diferenças entre níveis se tornem menos distintas à medida que se ganha mais experiência na preparação de software.

Cody, em seus artigos [COD82] e [COD74] identifica três níveis. O primeiro é o do desenvolvimento da matemática teórica ou análise numérica com seus métodos numéricos teóricos para resolver um particular tipo de problema. A ênfase neste nível está nas considerações teóricas, tais como análise do erro e provas de convergência.

O segundo nível da atividade é o da combinação de um ou mais métodos numéricos e sua transformação em algorítmo computacional para solucionar um determinado problema particular em um computador. O produto final deste segundo nível de atividade é muitas vezes considerado ser um programa computacional, especialmente quando o trabalho é publicado em uma linguagem algébrica de programação em um jornal profissional. Entretanto, usaremos o termo formal **algorítmo** para descrever o resultado final deste nível.

Cody define software computacional como sendo um módulo executável de programação, junto com a documentaçãoo dos algorítmos implementados em um ambiente específico.

A palavra-chave na definição de software computacional é implementação. Algorítmos na literatura não resolvem problemas em computadores, sómente implementações de algorítmos resolvem problemas. Um bom programa precisa ser baseado em um bom algorítmo, mas um bom algorítmo não garante um bom programa. Uma implementação imprópria de um ótimo algorítmo pode resultar numa péssima performance, comprometendo a reputação do algorítmo. Então, precisa-se atentar para os detalhes da implementação. Isto introduz o terceiro nível de atividade: a implementação de algorítmos para produzir um software computacional, isto é, o módulo executável de programação, junto com sua documentação.

Neste terceiro nível de atividade, a transformação de algorítmos computacionais em um software robusto freqüentemente envolve mudanças algorítmicas e inovações, as quais não podem ser feitas sem a ajuda da análise numérica.

O DESENVOLVEDOR DE SOFTWARE NUMÉRICO - O MATEMÁTICO COMPUTACIONAL

Existe uma significativa diferença entre o interesse e proximidade do analista numérico com a pessoa que desenvolve o software numérico. Esta pessoa será chamada de matemático computacional, para melhor caracterização.

O matemático computacional é um matemático que desenvolve, analisa e cria o algorítmo computacional, para obter uma solução aproximada para um problema matemático. Muitos destes problemas originam-se no campo de engenharia, da física, das ciências biológicas e sociais.

O matemático computacional tem como meta ser um suporte para outros trabalhadores e sua razão de ser é o sucesso do serviço numérico para usuários de computadores, através da criação e organização de bibliotecas de algorítmos numéricos. Eles não são, e nem pretendem ser analistas numéricos. Conquanto, o que eles são, essencialmente, numéricos fluentes, capazes de ler comandos em linguagem de alto nível, com uma apreciação por hardware e sistemas operacionais e devem ter a facilidade de escrever documentações claras e possuir habilidade de organização, somada à sensibilidade para as necessidades de usuários de computadores e ao desejo de vender seus produtos.

Os interesses do analista numérico são bem entendidos, entretanto, estão aquém das necessidades dos usuários de computadores. A força da motivação de um analista numérico é pessoal, estando muitas vezes tão envolvido em seu próprio interesse teórico, para produzir algorítmos ou programas úteis a um grupo de usuários.

Os trabalhos do analista numérico e do matemático computacional estão relacionados e não existe uma clara delimitação entre estas funções, além do que, existem muitos indivíduos, os quais atuam em ambas funções.

Duas atividades intelectuais de importância são a análise e a síntese. Analisa-se classes de problemas e classes de

métodos, mas depois, os resultados são de pouca utilidade para os analistas. Os resultados da análise precisam ser sintetizados em um programa. O matemático computacional é a pessoa interessada em ambas as atividades. E, segundo Gear, "a análise é a ciência e a síntese é a arte". O resultado será o software numérico propriamente dito.

Na aplicação da análise e síntese para produção do software numérico, o matemático computacional é, freqüentemente, requerido para desenvolver novos resultados matemáticos e para adaptar novos resultados para uso efetivo do computador digital automático. Os cálculos em algorítmos assim desenvolvidos são baseados tanto em considerações matemáticas, como em considerações computacionais.

O PROCESSO DE DESENVOLVIMENTO DO SOFTWARE

Para Johnston, em [JOH82], existem três passos básicos na criação de um software, os quais são:

a) A escolha do método básico a ser usado;
b) O projetar do algorítmo;
c) O produzir do programa como um item do software.

Ele define um **método** como uma fórmula matemática para achar a solução de um problema particular em estudo. Na escolha do método, é necessário que se tenha uma lista dos possíveis métodos a serem usados, o que implica numa pesquisa na literatura para se catalogar os existentes. Então, uma detalhada análise matemática de cada método necessita ser feita, para levantar princípios básicos, nos quais os métodos possam ser comparados, isto, juntamente com certa experiência computacional, a fim de avaliar o trabalho necessário para implementar cada método, extensões e comparações analíticas, para determinar o método ou combinação de métodos mais apropriada.

O segundo passo é o da descrição detalhada do processo computacional, que é o **algorítmo**, uma fórmula computacional projetada para resolver o problema com a exatidão quanto maior possível em um computador.

O passo final é o de converter o algorítmo em um iten do software. Este item do software é uma realização física do algorítmo, isto é, um **programa** do computador. Esta tarefa de produzir o programa, envolvendo diversas questões importantes e difíceis com respeito a estrutura aritmética usada na máquina, alocação de memória, velocidade dos dispositivos e operações do software. O objetivo deste passo é a plena utilização das capacidades computacionais disponíveis do sistema, de uma maneira mais eficiente possível.

Para Crowell e Fosdick, em [CRO77], o processo de produção de um software inicia com a análise algorítmica e prossegue através da construção do software e documentação, para vários testes e, finalmente, para a distribuição e suporte do produto do software. A exigência e o custo necessários são justificados pela utilidade de alta qualidade do software, pela eficiência do produto na sua distribuição e pelos benefícios providenciados à pesquisa em o aplicar.

Dew e James, em [DEW83], concordam com Crowell e Fosdick nas etapas do desenvolvimento do software. Eles partem, entretanto, de que, já se tenha os Métodos, não incluindo, portanto, a importante etapa da escolha do método básico a ser usado, descrita por Johnston. Eles caracterizam o desenvolvimento do software em cinco etapas, que são: o projeto e análise do algorítmo; a construção do software; uma documentação detalhada; vários testes e a distribuição e manutenção do software.

O projeto e análise de algorítmos

Na etapa do projeto e análise de algorítmos para desenvolvimento de um software, é onde são tomadas as decisões que influenciarão o software no que tange ao seu estilo, propósito e finalidade.

A produção de um software numérico não é uma simples extensão de programação básica e tem que ser bem mais do que um conjunto de programas que resolvem uma classe determinada de problemas. Programas que se pretende para distribuição pública necessitam delimitar o intervalo dos possíveis dados de entrada, de uma performance perante adversidades, compiladores e interpretadores, sistemas operacionais e características de hardware.

Nesta etapa é feita a definição de passos tomados para resolver o problema, são feitos convenções de notação, análises dos algorítmos, o relacionamento e o gerenciamento, compondo com estes elementos a filosofia e o estilo do software.

Esta etapa é uma das mais difíceis, pois, além do que já foi citado, é necessário um estudo dos métodos numéricos teóricos, para a tradução em algorítmos computacionais, o que muitas vezes requer reformulações nos métodos.

É ainda nesta etapa que se adota critérios, os quais são as características desejáveis de um software numérico, as quais são descritas mais adiante. Algumas destas são importante existirem em algorítmos a serem implementados, até para usuários sem maiores conhecimentos, tendo apenas a idéia básica do método. Estas são a **estabilidade**, o ser sensível aos erros de arredondamento, que são inevitáveis nos cálculos com números reais em computadores de precisão finita; a **exatidão**, que é a capacidade de resolver problemas com a exatidão pré-requerida na documentação, resolvendo corretamente a classe de problemas para que foi planejado, mesmo em caso de insucesso, com mensagens adequadas. Outra característica é a **eficiência**, pois quando implementado o algorítmo não deve gastar muito tempo ou espaço para armazenar ou resolver problemas.

Construção do software

A construção do software é a codificação do algorítmo, o verdadeiro código do algorítmo, incorporado no software, resultando num **programa e sua documentação**.

Nesta fase de transformação do algorítmo computacional no programa, o qual deve ter capacidade de detectar e contornar situações anormais, sem interromper a computação. O software deve conter facilidades para monitorar erros, a fim de determinar argumentos impróprios e prever problemas computacionais como underflow e overflow, antes que eles ocorram.

Esta transformação do algorítmo computacional num programa com as características acima, em geral, envolvem mudanças no algorítmo. O algorítmo alterado, então, deve ser analisado para garantir sua confiabilidade, pois uma implementação imprópria até mesmo de um magnífico algorítmo pode produzir resultados insatisfatórios, comprometendo a "reputação" do algorítmo. Ainda, um algorítmo óbvio não é, necessariamente, o melhor algorítmo e sua implementação em diferentes ambientes ou máquinas, produzem diferentes comportamentos.

Um esforço considerável é necessário para se construir um bom software, e dar a ênfase em ser capaz de transportar programas bem testados de uma instalação para outra, com mínimas modificações. Isto significa, que os matemáticos computacionais devem resistir a tentação dos programadores de explorar os recursos do sistema ou obscuras características especiais da linguagem de programação. Truques deste tipo produzem programas difíceis de serem compreendidos e podem falhar quando implementados em diferentes sistemas.

Deve-se projetar algorítmos com o propósito da adaptabilidade e codificá-los com a idéia de transportar o software de um computador para outro. Por exemplo, um teste do tipo:

" IF ABS(ERRO) < 0.5E-10 THEN ..."

não é adaptável, pois, devido a diferente precisão de cada computador, pode não ser possível obter a exatidão necessária em algumas máquinas. O programa, deste modo, irá se comportar diferentemente em diferentes sistemas, podendo falhar em alguns. Testes de erro devem ser expressos relativamente a precisão de cada computador. Uma melhor consideração para este teste seria definir no inicio do programa uma constante, por exemplo, "mi:=0.5E-10", que é a precisão do computador e usar nos testes:

" IF ABS(ERRO) < mi THEN ...".

Assim, só seria necessário modificar na passagem de um sistema para outro as constantes dependentes da máquina.

Portanto, num software numérico, alguns pontos, tais como testes de convergência, previsões de underflow e overflow, refletem o esquema de representação dos números um um computador. Estas porções do programa, que são dependentes da máquina, devem ser isoladas, tornando fácil sua alteração quando o programa é transportado para outra máquina.

Uma documentação detalhada

A documentação deve ser feita ao mesmo tempo que a construção do software, pois é parte deste produto final. A documentação influência o estilo do programa, o qual deve ser escrito para uma clara exibição da organização lógica da computação, localizando e bem definindo as partes dependentes do sistema.

A documentação precisa ser clara e bem pensada; organizada de maneira que permita um usuário ocasional obter informações necessárias facilmente.

A documentação deve ser consistente com as operações do programa, para que não haja surpresas no uso do programa.

Do ponto de vista do usuário, a documentação deve ser detalhada, contendo um enunciado clara da classe de problemas que o software, ou programa está designado a resolver, com todos os casos em que ele é particularmente aplicável; uma definição do tipo e propósito de cada parâmetro do programa; as restrições e a sua aplicação; um exemplo rodado de aplicação do programa, pois usuários irão freqüentemente procurar se basear no exemplo, adaptando-o ao caso particular que desejam resolver; deve conter uma avaliação de sua performance, isto é, informações detalhadas sobre velocidade de processamento e exatidão das respostas; enfim quaisquer características extras usadas no programa devem ser descritas e as principais seções dos programas devem ser destacadas pelo uso de comentários, dando uma estruturação clara, estando a listagem do programa disponível para quem quiser fazer modificações ou estendê-lo.

Resumindo, a documentação do software é a parte entre o que desenvolve e o usuário do produto e, por isto, deve ser completa, pertinente ao serviço a que se destina, com o nível adequado de detalhes. Quanto a que documentação produzir e como a produzir, deve-se ter em mente alguns princípios básicos:

a) Ela deve ser adequada e inteligível;
b) Deve-se testar a documentação do mesmo modo como se testa o próprio software;
c) Tem-se que escrever a documentação de modo a satisfazer seus leitores e usuários, não a você mesmo.

Vários testes

A determinação de testes numéricos nos programas matemáticos é uma tarefa complexa e muito demorada, que tem recebido muita atenção nos últimos anos.

Existem dois objetivos principais ao testar uma rotina. O primeiro é o de estabelecer experimentalmente o grau de correção do algorítmo. Para isto, seleciona-se um conjunto de problemas, chamados algumas vezes de "exercício de código", que são projetados para testar o maior número de possíveis caminhos do programa. O segundo objetivo é o de medir a performance

relativa do programa em comparação a outros. Uma vez assegurado que o programa é o algorítmo implementado, pois resolve o planejado, parte-se para a avaliação da performance. É importante entender que se está avaliando não o método numérico abstrato, mas o algorítmo implementado, o programa.

Quase toda avaliação de performance é feita pelo método de bateria, um grande conjunto de problemas para os quais o programa foi feito e preparado para resolver. A performance do programa é medida contra a de programas já existentes, registrando, por exemplo, o tempo requisitado para resolver cada problema, ou se foi convenientemente resolvido e quão exata foi sua solução.

Infelizmente, na maior parte das vezes, é difícil, mesmo após vários testes, dizer se um programa é absolutamente melhor que outro, uma vez que um programa pode resolver bem alguns problemas e outros não tão bem, enquanto que o oposto pode acontecer com outro programa. Esta é a razão de existirem bibliotecas de softwares, contendo vários programas para resolver um mesmo problema.

Distribuição e manutenção do software

Nesta etapa, pode-se justificar todo custo e esforço despendido para o desenvolvimento do software, através da distribuição do software, que pode ocorrer através de sociedades profissionais de suporte, como a "Association for Computing Machinery" (ACM), com seus grupos especiais de interesse em matemática aplicada ou numérica, como SIGMAP e SIGNUM, para os quais são enviados artigos, onde são analisados e divulgados. No Brasil, temos a Sociedade Brasileira de Matemática Aplicada e Computacional (SBMAC), que tem se preocupado em organizar uma programoteca da sociedade, e de divulgar softwares a ela enviados através de seu Boletim. Existe também a Sociedade Brasileira de Computação (SBC) com propósitos semelhantes.

A divulgação dos softwares está associada a ênfase que o software incorpora, que podem ser utilidade, explorável na pesquisa ou útil no melhoramento da produção de software.

Os projetos de software, em geral, originam-se do esforço de centros universitários, de laboratórios governamentais e de empresas privadas.

O importante, quando se desenvolve um software, é assumir a responsabilidade da manutenção do produto, incluindo correções de erros, modificações, extensões e o suprir novas necessidades do usuário.

Software numérico de boa qualidade enfrentam a expectativa de usuários, que tem sido significativamente mudada no último quarto de século, incluindo atividades de facilidades de uso. Um programa, o qual é o mais rápido e mais exato, poderá ser ignorado, se existir mesmo um número pequeno de impedimentos de uso. Os desenvolvedores de softwares comercial entenderam isto, mas muitos pesquisadores não, e ainda desprendem grande esforço para produzir um programa encorporando o melhor algorítmo, somente para ver o fruto de seu laborioso trabalho ignorado.

CARACTERÍSTICAS DESEJÁVEIS DE UM SOFTWARE NUMÉRICO

O propósito de desenvolver programas de um software numérico é providênciar ferramentas computacionais úteis para programação científica, mas, para isto, estas rotinas necessitam reunir certas características, critérios considerados essênciais, que são exatidão, eficiência, confiabilidade, validação, facilidade de uso, boa documentação, flexibilidade, modificabilidade, robustez e transportabilidade.

Muitas destas características já foram citadas, exemplificadas nas seções anteriores, mas para uma melhor formalização e caracterização, serão redefinidas, apesar da dificuldade, da dependênciaa e interrelacionamento destas características.

Exatidão e eficiência

Apesar de os critérios de um bom software tem sido mudados com a experiência e tecnologia, a exatidão e eficiência têm se mantido.

Exatidão é a capacidade do programa resolver problemas com a exatidão pré-requerida na documentação.

O critério de eficiência mudou com o desenvolvimento. No princípio, quando os computadores tinham pouca memória e esta era muito cara, um programa eficiente era um que usava a memória eficientemente, sendo o menor possível. Hoje, a memória está tão disponível e barata, fazendo com que este conceito evoluisse para velocidade de processamento; e as máquinas tornando-se cada vez mais velozes, este componente de eficiência deixou de ser tão importante.

Confiabilidade e validação

Confiabilidade é a característica que garante a consistência da operação do programa com sua documentação, resolvendo corretamente a classe de problemas para que foi planejado. Resolver corretamente refere-se a habilidade do programa de obter resultados numéricos desejados eficientemente. Nos casos em que não conseguir encontrar a solução, deve finalizar com uma mensagem de erro adequada.

Validação é um critério importante na produção de um software. O propósito da validação é estabelecer que a rotina cumprirá seu objetivo. Para isto, de inicio, necessita-se estar claro de seu objetivo, da classe de problemas que podem ser resolvidos e o que se entende pela "solução".

O processo da validação de um programa inclui uma detalhada análise matemática do algorítmo, uma prova de que o programa é uma representação fiel do algorítmo e, finalmente, um extensivo teste com representativo conjunto de problemas para testar a maior parte de possíveis caminhos do programa.

Uma apreciação imprópria do ambiente computacional durante a implementação pode degradar a confiabilidade e validação, restringindo o conjunto de problemas que podem ser resolvidos.

Facilidade de uso

O critério facilidade de uso está associado a ter boa documentação, mas também, quanta informação é requerida ao usuário para o uso de tal programa. Um programa fácil de ser usado é aquele que somente requer a entrada (dados iniciais), definições do problema a ser resolvido. Qualquer análise do problema para o propósito de setar os parâmetros do método é feito automaticamente pelo programa.

Um software fácil de ser usado é o que antecipa todas as opções que todos usuários possam querer usar, através de um controle que seleciona a opção desejada.

Boa Documentação

A documentação de um programa deve ser clara, conscisa, fácil de entender, explanando como funciona e como usar. Isto é útil não somente para decidir se, usa-se ou não o programa, como também, assistir no diagnóstico de cada dificuldade proveniente do uso. Ela deve ser organizada de maneira que permita um usuário ocasional obter informações necessárias facilmente.

Mensagens de erro são necessárias diagnosticando o acontecido, e sua gravidade. Um exemplo, numa rotina para resolver equações não lineares, pode não ser possível encontrar uma solução com a precisão requerida pelo usuário, em tal caso, a rotina deve apresentar a melhor solução que pode encontrar, dando o número de iterações e a exatidão contida em tal resultado, mais uma indicação de que não foi conseguido a exatidão requisitada.

A documentação vai além de manuais, incluindo a organização de resultado, de forma a documentar resultados, e ainda, a documentação do código, de forma a ser fácil de ser compreendido par manutenção, atualização, alterações e extensões.

Flexibilidade e modularidade

A característica de flexibilidade está associada a característica de facilidade de uso, permitindo que o usuário modifique valores de parâmetros, condições iniciais ou até mesmo de método de resolução do problema.

A modularidade é outra característica muito importante, e influência o estilo da programação, pois separa as tarefas a serem efetuadas em partes, denominadas módulos, o que gera um programa estruturado, facilitando o entendimento e manutenção do software. A modularidade é muito importante para garantir a transportabilidade de um software, uma vez que existem partes dependentes de máquina que necessitam ser modificadas, adaptadas ao novo ambiente. Estas partes são isoladas e organizadas de maneira que se possa modificar facilmente; esta parte deve ser bem identificada na documentação.

Dentro do conceito da modularidade, alguns autores introduzem a modificabilidade, que é a facilidade de se modificar e adaptar o software a um problema específico que se quer resolver. Esta questão é abordada por R. C. Bushnell, no artigo denominado: "User modifiable software" (em [RIC71], seção 5.2, p.59-66).

Robustez

A robustez é um indicador da extensão da rotina. Um método numérico envolve alguns parâmetros, os quais podem assumir valores determinados de acordo com as propriedades básicas do problema particular. A análise do processo é geralmente baseado no comportamento do problema modelo. O comportamento da maioria dos problemas não divergem, na prática, do esperado pelo modelo. Mas existem problemas em que a solução se deteriora, afastando-se do modelo. A extensão deste afastamento é uma medida de robustez, da potencialidade do programa.

Robustez se refere também, a habilidade do programa de contornar dificuldades, sem, necessariamente, interrupção do programa. Quando a interrupção se faz necessária, devido a erros, o software tem que providenciar um diagnóstico preciso do erro e sua gravidade. O problema de underflow é um exemplo. Usualmente, quando ocorre, dá-se uma mensagem de erro e considera-se o valor com zero. Se o underflow influenciar consideravelmente o resultado computado é chamado de "destrutivo", caso contrário é chamado de "não destrutivo". O problema com a mensagem de underflow é que ela não distingue entre o caso destrutivo e o não destrutivo.

Idealisticamente, deve-se reestruturar o programa de modo a evitar underflow; caso ocorra o underflow não destrutivo, o valor é substituído por zero, e caso ocorra o underflow destrutivo, deve ser dada uma mensagem de erro, diagnosticando precisamente o fato.

Transportabilidade

Transportabilidade envolve os conceitos de portabilidade e adaptabilidade. Portável é quando o programa pode ser executado corretamente em um novo ambiente sem nenhuma espécie de mudança, o que é, em geral, idealístico na prática. Se um programa pode ser transportado a outro ambiente, mantendo sua performance, com apenas poucas modificações bem documentadas, ele é adaptável ou transportável.

Estas características são relevantes a mudanças na performance do programa quando o ambiente da máquina é mudado. Isto é uma questão muito difícil, porque existem muitos fatores, os quais afetam a performance.

Idealisticamente, um software numérico deveria ser completamente portável, mas, devido as diferenças de máquinas e seus ambientes, é praticamente impossível atender este objetivo. Existem, entretanto, algumas medidas que podem ser tomadas para ajudar o grande tratamento da uniformidade da performance.

Uma das causas das diferentes performances é a variedade de arquiteturas de máquinas existentes. O remédio é fazer várias versões de rotina, uma para cada tipo de máquina. Isto não é portável, no sentido da palavra, é adaptável; mas o é para usuário.

Outra atitude para eliminar diferenças na performance é escrever o programa num dialeto "standart" da linguagem de programação, sem explorar características especiais de implementações dependentes de máquina.

Outra atitude, ainda, é definir valores que são dependentes da aritmética da máquina no inicio do programa, como constantes, e no transporte de uma máquina para outra, estes valores devem ser redefinidos de acordo com a aritmética da máquina, para evitar que se tente calcular exatidões maiores do que se pode calcular.

A INFLUÊNCIA DO TIPO DE USUÁRIO NO DESENVOLVIMENTO DE SOFTWARE

O software numérico se consistui não apenas de rotinas, mas é um programa que gerência várias rotinas numéricas, que resolvem certa classe de problemas.

A abrangência do software numérico é decorrente do tipo de usuário a que se destina, o que é manifesto pelo tipo de instituição que custeam o seu desenvolvimento, as quais, em geral, são Universidades, laboratórios governamentais e empresas privadas, individualmente ou em cooperação entre elas.

Os usuários de software influênciam o desenvolvimento de softwares, pois quando se desenvolve um software para um grupo de usuários, há limitações na diversidade de interesses e motivações que podem ser abrangidos, devido a conflitos entre conjuntos de interesses, como: entre a pesquisa e o ensino; entre a pesquisa geral e a pesquisa especializada; e entre usuários de uso geral e analistas numéricos.

Um software instrucional, como uma biblioteca de ensino, requer que métodos tradicionais estejam incluídos, uma vez que são ensinados, enquanto que uma biblioteca de pesquisa de uso geral ou um software aplicativo somente requer os métodos considerados como melhores em certa área. Num software instrucional métodos ruins são necessários para que se possa fazer comparações com os bons. Em softwares aplicativos são necessários resultados com grande exatidão, e calculados otimizadamente, enquanto que, em softwares instrucionais, a visualização de cálculos passo a passo são úteis a compreensão do método e a verificação de sua convergência.

Software aplicativo, com finalidade de pesquisa geral ou pesquisa especializada, gera outro conflito, pois grupos de pesquisa especializada, por causa da penetração em áreas numéricas particulares, requerem uma abrangência muito maior nestas áreas do que os de finalidade de pesquisa geral.

O próprio comportamento de analistas numéricos que, em geral, efetuam mudanças de conteúdos, códigos e na própria documentação, costumeiramente faz com que o software tenha características de modularidade, para que seja fácil de fazer

alterações e expansões; enquanto que para usuários de uso geral, que apenas utilizam o software de uma forma "estática", ou seja, o software e a documentação permanecem os mesmos para sempre. Isto requer um software fácil de ser usado, com boa documentação e robusto.

A documentação também deve ser adequada a finalidade do software, tanto em qualidade como em quantidade. Para software aplicativo comercial, J. D. Lomax em seu livro "Documentação de software" (LOM83]), descreve a documentação em três categorias:

a) Informações preliminares;
b) Documentação do usuário;
c) Documentação de manutenção.

costituindo uma "grande coletânea" de documentação.

Para software aplicativo científico, pode ser simplificado e minimizado em documentação do usuário e documentação do programa.

Para software instrucional, a documentação é ainda compensada com um estilo conversacional nos programas, permitindo ao usuário fazer inúmeras variações nos dados e parâmetros de entrada. Desta forma, para um mesmo problema o usuário pode, facilmente, alterar um dos parâmetros e observar o efeito da mudança nos resultados, o que lhe permite um controle do comportamento da resolução.

O estilo conversacional proporciona maior interação do usuário com a máquina, dando a ele uma maior clareza dos passos que estão sendo efetuados no processamento, proporcionando, ainda, maior flexibilidade no uso do software, amenizando o "efeito místico", que era resultante de software "caixa-preta", dando assim, maiores informações.

O estilo conversacional proporciona ainda um software fácil de ser usado, desde que em cada nível de tarefa possa ser caracterizado por telas instrucionais que antecipem todas as possíveis tarefas que todos os usuários possam querer requerer. O programa neste estilo deve ser auto-documentado, para proporcionar esclarecimentos a usuários menos instruídos, e que as perguntas sejam claras e definam bem a resposta, não permitindo ambigüidades. Uma técnica usada é chave de seleção, onde na tela estão todas opções possíveis, e o usuário escolhe uma delas, através de uma letra, um número ou uma tecla. Isto também é conhecido como menu.

SINAI-16 UM EXEMPLO DE SOFTWARE INSTRUCIONAL

O SINAI-16 (Software Interativo Numérico Aplicativo e Instrucional em micros de 16 bits) foi desenvolvido no Laboratório de Programação do Instituto de Informática da PUCRS, com a finalidade de auxiliar o ensino de cálculo numérico e métodos computacionais e também servir de suporte a pessoas que em seus trabalhos necessitam resolver equações algébricas e transcendentes.

Em seus seus quase seis meses de disponibilidade, foi utulizado por diversas turmas de cálculo numérico, facilitando a

aprendizagem e aumentando o interesse dos alunos pelo assunto. Ele também foi eficiente na resolução de alguns problemas dentro da comunidade científica.

O SINAI-16 proporciona a resolução de equações polinomiais de grau menor ou igual a vinte, calculando as raízes reais e complexas, inclusive reais múltiplas. Proporciona, ainda, um módulo que possibilita traçar gráficos e imprimi-los (caso se tenha impressora gráfica). Ele contém um estudo completo deste capítulo numérico; incluindo a localização de raízes, enumeração através da visualização gráfica e em alguns casos, através de resultados obtidos, a determinação de valores iniciais para métodos iterativos; o cálculo de raízes pelos métodos tradicionais e por alguns outros de importância científica atual; e até a deflação de raízes pelo método de Briot-Ruffini.

Este pacote é interativo, proporcionando que a qualquer momento se possa modificar ou corrigir valores dos coeficientes; ou trocar a estimativa inicial, ou a exatidão desejada, e, até mesmo, trocar o método de resolução, sem que tenha que entrar novamente com os coeficientes da função.

O SINAI-16 é um pacote fácil de ser usado, devido ao estilo conversacional, e auto-documentado por tela instrucionais, contendo menus de seleção e telas de orientação e instrução, além de alguns efeitos sonoros para tornar o primeiro contato mais agradável.

A documentação é adequada ao estilo, tendo as opções de uso, antecipadas por menus, que compõem as telas instrucionais, as quais facilitam o uso do software, pois não há necessidade de conhecimento prévio para utilização do software. Além disto, os manuais são organizados de modo a conceder a informação desejada, por qualquer usuário, a qualquer momento.

Métodos implementados

Os métodos implementados na primeira seção do SINAI-16 (resolução de polinômios), visaram atender finalidades aplicativas e instrucionais, incluindo métodos que são usualmente ensinados nos cursos de cálculo numérico e métodos computacionais, bem como métodos novos ou mais eficientes e complexos que têm seu valor científico.

Capacidade Gráfica: O SINAI-16 pode traçar gráficos de polinômios de grau até vinte, com intervalos de definição variável, capaz de auxiliar a determinação da enumeração e separação de raízes.

Cálculo de Cotas: Calcular cotas, ou intervalos que contenham todas as raízes da equação, além de proporcionar boas estimativas de vlores iniciais para métodos iterativos. As cotas implementadas foram a de Cauchy, que, em geral, está próxima a maior raiz em módulo; e a Fase Inicial, que determina um círculo que contém no mínimo uma rais, estimando um valor inicial próximo a menor em módulo, além de identificar a existência de raízes complexas.

Cálculo de raízes reais de equações: Os métodos implementados foram a da Bissecção; de Newton-Rapshon; da Secante; de Newton para raízes múltiplas; o método Híbrido C e o método MIDREM.

Cálculo de raízes complexas de equações: Os métodos implementados para o cálculo de raízes complexas foram o método de Newton para complexas e o de Bairstow.

Deflacionar de raízes: As raízes calculadas reais ou complexas podem se deflacionadas, reduzindo o grau do polinômio, através do método de Briot-Ruffini e algorítmo de Horner.

Documentação

Por ser um software conversacional, o SINAI-16 é auto-documentado, contendo em seções principais telas de informações e de auxílio, descrevendo as funções implementadas, para orientar ao usuário no momento da execução e utilização do programa, bem como para instruir o usuário, suprindo assim, as finalidades aplicativas e instrucionais.

Além disto, o SINAI-16 possui dois manuais: o **Manual Instrucional do Usuário (MIU)**, que contém um conjunto completo de informações do ambiente computacional, de instalação, organização e utilização do software. Neste último item, destacam-se as fichas instrucionais que além de exemplos para ilustrar a utilização do software, contém todas as informações para pessoas que queiram utilizar métodos de resolução de equações tanto com ênfase da análise numérica; da prática de utilização do método; ou da ênfase computacional. O outro manual é denominado **Manual de Manutenção do Programa (MMP)**, que além de um conjunto básico comum ao MIU, contém informações para garantir a transportabilidade do software a outro equipamento, incluindo a listagem completa do software.

Outro módulos

Está sendo feito a seção de resolução de equações algébricas e transcendentes, estando esta em fase de depuração e ajustes finais na documentação. Esta nova seção permite traçar gráficos de equações trancendentes e o cálculo de raízes pelos métodos da Bissecção; Newton-Raphson; Secante e pelo Híbrido C.

Maiores informações podem ser obtidas na PUC - RS INFORMÁTICA (Prédio 30), caixa postal 1429, cep 90620 -Porto Alegre - RS, aos cuidados do Projeto SINAI-16.

Referência Bibliográfica

[COD74] CODY, W. J. The construction of numerical subroutine libraries. SIAM review, Philadelphia, 16(1):36-46, Jan.1974.

[COD82] CODY, W. J. Basic concepts for computational software. In: INTERNATIONAL SEMINAR ON PROBLEMS AND METHODOLOGIES IN MATHEMATICAL SOFTWARE PRODUCTION, Sorrento, Nov.3-8, 1980. Proceedings. Berlin, Springer-Verlag, 1982. p.1-23.

[CRO77] CROWELL, W.R & FOSDICK, L.D. Mathematical software production. IN: MATHEMATICAL SOFTWARE III. New York, Academic Press, 1977.

[DEW83] DEW, P.M & JAMES, K. R. Introduction to numerical computation in PASCAL. Hong Kong, Macmillan Press, 1983.

[DIV86] DIVERIO, T. A. Software Numérico Aplicativo e Instrucional. Porto Alegre, CPGCC da UFRGS, 1986. (dissertação)

[DIV86a] DIVERIO, T. A. SINAI-16 Manual instrucional do usuário. Porto Alegre, CPGCC da UFRGS, 1986.

[DIV86b] DIVERIO, T. A. SINAI-16 Manual de manutenção do programa. Porto Alegre, CPGCC da UFRGS, 1986.

[GEA80] GEAR, C. W. Numeraical Software: science or alchemy?. Advances in computer, New York, 19:229-48, 1980.

[JOH82] JONHSTON, R.L. Numerical methods a software approach. New York, John Willey & Sons, 1982.

[LOM83] LOMAX, J. D. Documentação de software. Rio de Janeiro, Campus, 1983.

[RIC71] RICE, J. R. Mathematical software. New York, Academic Press, 1971. (ACM Monograph series)

EL LENGUAJE DECLARATIVO COMO HERRAMIENTA DE IMPLEMENTACION:

Un Caso de Estudio en Telecomunicaciones

J. Grompone, F. Brum, M. Casamayor, F. Simini

INTERFASE LTDA.
Zabala 1378, Montevideo, Uruguay
Teléfono: 95 31 66, 95 62 65

RESUMEN

El presente trabajo resume una experiencia de Diseño e Implementación de una Central de Conmutación Télex.

Se presenta y discute la alternativa de implementar un Lenguaje Declarativo como solución efectiva, tanto para la Operación del Sistema, como desde el punto de vista de la Ingeniería de la Programación.

Se comparan las diferentes soluciones a través del análisis de las prestaciones, de los tiempos de ejecución y de los tiempos de desarrollo, concluyendo que la construcción de una herramienta tan abstracta y conceptual como un Lenguaje, es también una solución eficiente para problemas concretos de implementación.

1. INTRODUCCION

En este trabajo se presenta uno de los resultados de un proyecto de diseño e implementación de un Conmutador Télex: una manera general de resolver el problema de personalizar la conmutación. Las ideas desarrolladas no están limitadas al tema Télex sino que pueden generalizarse a diversos problemas de telecomunicaciones y de otras áreas.

La personalización de un Conmutador de Datos exige definir todos los elementos potencialmente modificables del equipo:

- cantidad y propiedades de las líneas
- manera como se agrupan las líneas
- enrutamiento de llamadas
- servicios y prestaciones auxiliares

El enrutamiento es el procedimiento por el cual se asocia a un código de selección de abonado una línea para realizar una comunicación.

El problema del enrutamiento fue analizado desde tres puntos de vista:

a) Definir e implementar los algoritmos y las estructuras de datos capaces de enrutar una solicitud de conexión en tiempos aceptables para el correcto funcionamiento

del Conmutador.

b) Lograr que la programación del Conmutador fuera independiente de su personalización.

c) Dotar al personal de operación del equipo de herramientas flexibles para introducir modificaciones y personalizarlo de acuerdo a necesidades cambiantes.

La solución adoptada consistió en desarrollar un Lenguaje Descriptivo que permite definir con precisión las características del Conmutador. Dicho lenguaje fue llamado Lenguaje de Especificación de Personalidades (de ahora en más LEP). Paralelamente al LEP, se desarrollaron Autómatas que interpretan las tablas generadas por el compilador LEP, realiza el enrutamiento y algunas otras funciones y un Simulador utilizado como herramienta previa a la instalación de una Personalidad.

En este trabajo se presentan algunos de los resultados extraídos de la etapa de diseño, implementación y explotación del LEP. El artículo se concentra en el tema de diseño de un Lenguaje como solución de Ingeniería para problemas con las características del que enfrentamos. Este último punto llevó a confrontar el enfoque adoptado con el enfoque convencional (construcción de un conjunto de programas que implementen en concreto cada Plan de Enrutamiento sin proporcionar una visión abstracta y global del mismo).

En este trabajo, además de presentar ambas soluciones y la discusión correspondiente, se incluyen los resultados medidos a lo largo del Proyecto. La utilidad del LEP fue considerada desde dos ópticas:

a) Para las tareas de operación (una vez instalado el Conmutador) es una herramienta poderosa que simplifica ampliamente su gestión y todos los aspectos vinculados al mantenimiento.

b) Desde el punto de vista de la duración del proyecto en su conjunto, los resultados obtenidos muestran que el tiempo invertido en el desarrollo del LEP fue compensado ampliamente por el ahorro obtenido al no tener que implementar varias versiones concretas de Planes de Enrutamiento.

2. EL PROBLEMA A RESOLVER

Se llama Personalidad de un Conmutador al conjunto de tablas y de programas que implementan las características operativas de un equipo dado. En la Personalidad se definen

las modalidades de las líneas (abonado o troncal, etc), las reglas de enrutamiento (numeración, rutas, etc), las reglas de accesibilidad entre líneas y servicios y nombres simbólicos accesibles desde la Consola (que permiten ajustes y monitoreo en tiempo de operación).

Para el Conmutador una llamada es una solicitud de conexión. La función básica de la Personalidad es asociar a un destino a una llamada. La noción de destino encierra diferentes contenidos:

- la solicitud no es posible
- es una línea (o un haz) de salida para una comunicación télex
- es un mensaje aclaratorio
- es un servicio interno del Conmutador

También es necesario que el Conmutador suministre información a los efectos de la tarifación, por lo que la Personalidad debe poder identificar al abonado llamante (aquel que corresponde a la línea por la que llega la llamada).

Del análisis de las funciones de la Personalidad surge la identificación de cuatro tareas independientes:

- Enrutamiento: asociar un destino a una llamada.

- Verificación de accesibilidad: determinar si la línea llamante tiene derecho a conectarse con el destino solicitado.

- Identificación del abonado correspondiente a la línea llamante.

- Conversión de códigos: en el caso de que el destino sea un nuevo Conmutador (por ejemplo una llamada internacional).

De acuerdo a esto, el paquete de la Personalidad consta de cuatro procedimientos bien definidos. A los efectos de este trabajo interesa considerar las diferentes alternativas para su implementación y discutir las ventajas y desventajas relativas a cada una de ellas.

3. VARIANTES DE IMPLEMENTACION

Segun la experiencia de años de diseño de conmutadores es posible distinguir diferentes niveles de implementación de una Personalidad [1], [2]:

En el primer nivel se diseña una programación dedicada,

escrita en Assembler, y alojada en ROM. En este caso el mantenimiento de la personalidad no es simple.

En el segundo nivel de implementación, la programación está escrita en "C" pero reside por igual en ROM. Se mejora la posibilidad de mantenimiento, pero al igual que en nivel anterior, el personal de operación del equipo no puede realizar modificaciones que no impliquen la reprogramación de las EPROMs.

En el tercer nivel de implementación, reside en ROM un conjunto de programas interpretadores de tablas y la Personalidad consiste en un conjunto de tablas que constituyen la entrada de los algoritmos residentes. El mantenimiento queda en manos del personal de operación del equipo, y se resuelve el problema de la independencia. Sin embargo, la tarea de modificación de las tablas es tan compleja que resulta inviable.

En el cuarto nivel, las tablas de entrada de los algoritmos residentes son generadas en forma automática por un Compilador a partir de un programa escrito en el lenguaje declarativo LEP. Esta opción deja en manos del personal de operación del equipo la responsabilidad del mantenimiento proporcionándole herramientas que hacen posible su tarea. La descripción de la Personalidad en LEP constituye además un resumen global de un alto nivel de abstracción que facilita el trabajo tanto del personal de operación como del equipo de diseño.

Estos cuatro niveles se resumen en dos opciones:

- Considerar a cada Conmutador concreto como un caso único, escribiendo programas específicos que implementan su Personalidad.

- Construir un Conmutador genérico, donde la Personalidad sea suministrada desde la Consola mediante tablas. Esta opción implica, debido a la complejidad de las tablas, la construcción de un Lenguaje Declarativo. El LEP es imprescindible en esta opción ya que el mantenimiento manual de las tablas es inviable.

El proyecto consistió en la implementación de un modelo de Conmutador de 512 líneas y la instalación de cuatro ejemplares con diferentes personalidades. Se eligió la solución LEP con los siguientes objetivos:

- Independencia entre la programación y la entonación. Los ajustes de la Personalidad son realizados por el personal de operación del equipo, resultado importante si tenemos en cuenta que por ejemplo el Conmutador Internacional

de nuestro Pais se encuentra en este momento en la versiòn 36 de su Personalidad.

- Obtenciòn de una herramienta descriptiva general utilizable en otros modelos de Conmutadores. Un programa LEP describe completamente un Conmutador, proporcionando un lenguaje preciso compartido por los usuarios y el equipo de diseño.

El diseño y la implementaciòn del compilador LEP fue realizado en paralelo con la implementaciòn de personalidades dedicadas para los primeros ejemplares del Conmutador que entraron en funcionamiento. Esto es que pese a optar por la soluciòn LEP, por razones de plazo de entrega, tambièn fue necesario utilizar la soluciòn de programaciòn dedicada. El desarrollo de esta experiencia en paralelo permitiò confrontar ambas aproximaciones no solamente desde la òptica de las prestaciones, sino tambièn comparando los tiempos invertidos en cada alternativa.

4. EL LENGUAJE DECLARATIVO LEP

El lenguaje LEP se diseñò con dos objetivos a la vista:

- tratar que fuera simple y natural en su expresiòn

- adherir a las recomendaciones del Comite Consultivo Internacional para la Telefonia y Telegrafia (CCITT) para el lenguaje hombre-màquina [3].

A los efectos de ilustrar el lenguaje declarativo definido, se presenta, a modo de ejemplo, un programa simple:

```
plan  DEMO  D'16     /* Programa de Demostraciòn */

      /* Declaraciòn de reglas de enrutamiento */
      rule
           TELEX;
      end rule

      /* Declaraciòn de servicios de consola */
      features
           CONSOLA-AUX        { BAU:L=@ };
           TEXTO-ABC          { TEP:ABC:L=@ };
      end  features

      /* Declaraciòn de mensajes */
      messages
           NCH     "<^NCH<^";
      end  messages
```

```
/* Declaración de formatos */
recode
     TAL-CUAL            *      *;
     SACA-1-CIFRA      ?*       *;
end  recode

/* Declaración de haces de lineas */
group

     ABONADOS
          ( O'0..O'11 )
     TELEX multiple trunk Akey WRU ;

     TRONCAL
          ( O'12..O'17 )
     TELEX single terminal Akey log bill WRU ;

end group

/* Declaración de permisos */
forbidden
     ABONADOS  CONSOLA-AUX;
     TRONCAL   TRONCAL, CONSOLA-AUX;
end  forbidden

/* Numeración externa de las líneas */
identification
     ABONADOS +      10[0..9];
     TRONCAL         000;
end  identification

/* Plan de enrutamiento */
routing   TELEX
     000             CONSOLA-AUX;
     0[2..9]*        TRONCAL SACA-1-CIFRA;
     10[0..9+]       ABONADOS;
     115             TEXTO-ABC;
     20?             NCH;
end  routing

end  plan
```

El elemento básico de la Personalidad es el Plan de Enrutamiento. Este Plan se define como una secuencia de declaraciones de la forma:

código de selección destino

Los códigos de selección se describen o bien completamente (P. Ej. 000) o bien utilizando abreviaciones (P. Ej. 0[2..9]* que representa los códigos 02, 03, hasta 09 seguidos de otros dígitos).

El objetivo màs frecuente de una llamada es establecer una comunicaciòn con otro abonado a travès de una lìnea. El LEP permite describir completamente una lìnea indicando sus atributos. Para ello las lìneas se agrupan en haces. Un haz es un conjunto ordenado de lìneas designado por un identificador. Los haces son particiones disjuntas, ordenadas y completas del conjunto de las lìneas.

En la secciòn group se asignan nombres a los haces.

En la secciòn recode se asignan nombres a funciones de recodificaciòn de còdigos utilizadas cuando el destino es un nuevo Conmutador. La clausula:

```
0[2..9]*    TRONCAL    SACA-1-CIFRA
```

indica que si por ejemplo se disca 05247, que tiene al haz TRONCAL como destino, el còdigo que se retransmite es 5247.

El LEP permite definir otros destinos mediante las siguientes secciones:

- features: se asignan nombres a los servicios provistos por la Consola (P. Ej. TEXTO-ABC se usa para designar el servicio de texto de prueba accesible a los abonados). De acuerdo con los ejemplos, la clausula:

```
115    TEXTO-ABC
```

indica que al discar el 115 el abonado obtendrà el texto de prueba.

- messages: se asignan nombres a los mensajes provistos por el Conmutador. La sentencia:

```
20?            NCH
```

indica que al discar cualquiera de los còdigos de tres dìgitos que comienzan con 20 se obtendrà el mensaje:

```
<CR> <LF> NCH <CR> <LF>
```

Como puede observarse, un programa LEP està formado por declaraciones que describen o bien atributos de los elementos primitivos de nuestro problema de enrutamiento (lìneas, mensajes, etc.) o bien relaciones entre los elementos primitivos que permiten agregarlos en estructuras màs complejas.

El compilador traduce la descripciòn de alto nivel en una serie de tablas que a su vez son utilizadas por los algoritmos

residentes que conforman el núcleo de la Personalidad.

La operativa es la siguiente: el personal de operación del equipo escribe su programa LEP (o modifica uno existente), lo compila, verifica sus prestaciones utilizando un Simulador y posteriormente lo instala mediante una orden de Consola. Durante la instalación de una nueva Personalidad en un Conmutador se interrumpe el enrutamiento (no se realizan nuevas conexiones) pero las comunicaciones ya establecidas se continuan sin cambio.

Los puntos analizados en este ejemplo constituyen una aproximación ilustrativa a las características del LEP. La descripción detallada del lenguaje aparecerá en publicaciones del área de Telecomunicaciones.

5. IMPLEMENTACION DEL COMPILADOR

El Compilador LEP se implementó en forma clásica [4], mediante cuatro módulos independientes:

- análisis Lexicográfico
- análisis Sintáctico
- generador de Tablas
- verificación de Consistencia Semántica

Los diferentes módulos fueron escritos en "C", a partir de la definición del lenguaje. Inicialmente se escribió un prototipo del analizador sintáctico en Prolog.

6. COMPARACION DE LOS TIEMPOS DE EJECUCION

Para el modelo de Conmutador de 512 líneas existen dos modalidades de implementación de personalidades:

- las implementaciones dedicadas
- las personalidades compiladas utilizando el LEP

Es importante comparar ambas modalidades desde el punto de vista del tiempo de ejecución ya que éste es un aspecto crítico para el funcionamiento de un Conmutador.

Se compararon 15 casos de enrutamiento, comprendiendo tanto casos típicos como casos límite. Los tiempos fueron medidos utilizando un Simulador. Como resultado de esta comparación se tienen siete casos en que la versión LEP demora más que la programación en C dedicada, cuatro casos en que demoran lo mismo, y cuatro casos en que es mejor.

Tomando los tiempos globales se tiene una pequeña

desventaja en tiempos con el LEP. Se debe destacar, sin embargo, que los cuatro interpretadores de tablas poseen todas las protecciones necesarias para asegurar que terminan su ejecución en un tiempo adecuado en todos los casos. Estos resultados permiten afirmar que no hay diferencias apreciables entre el funcionamiento de Conmutadores de uno y otro tipo.

Desde el punto de vista de la operación es claro que el modelo LEP supera ampliamente al modelo de programación dedicada.

7. COMPARACION DE LOS TIEMPOS DE DESARROLLO

Se verá que -contra lo que puede pensarse a priori- el esfuerzo necesario para implementar la solución LEP no es superior al de implementar los Conmutadores con programación dedicada, aún cuando el número de ejemplares a ser instalado sea reducido (en nuestro caso de cuatro). Además, la solución LEP proporcionó, como resultado indirecto, una herramienta de especificación e implementación muy valiosa para proyectos futuros.

Los cálculos de tiempos se realizan sobre la base de la siguiente igualdad:

1 mes-hombre = 170 horas de trabajo realizado

El replanteo de las horas-hombre empleadas en el proyecto en su conjunto conduce al siguiente cuadro de resultados:

ETAPA	HORAS	MESES	%TOTAL
diseño del LEP	410	2.4	5%
preparación programadores	560	3.3	7%
especificación	720	4.2	9%
implementación	4764	28.0	62%
puesta en marcha	1231	7.2	16%
TOTAL	7685	45.2	

El resumen global, dejando de lado el tiempo de formación de programadores, es el siguiente:

ETAPA	MESES-HOMBRE	% TOTAL
Diseño	6.6	15%
Implementación	28.0	67%
Puesta en marcha	7.2	17%
TOTAL	41.8	

El rendimiento de un mes-hombre a lo largo del Proyecto se ajustó notablemente a las estimaciones teóricas (según las cifras del metodo de estimación COCOMO se obtienen 400 líneas/mes-hombre para este tipo de proyectos) y se presenta en el siguiente cuadro:

LENGUAJE	PAGINAS	LINEAS	TIPO
Assembler (*)	107	5.300	Final
C	150	7.500	Final
Assembler	10	500	Auxiliar
C	54	2.700	Auxiliar
ADA	14	700	Auxiliar
Total Final		12.800	
Total Auxiliar		3.900	23% del total
TOTAL		**16.700 líneas**	

(*) Se tomó 50% de la programación total (10.600 líneas) como programación nueva ya que se reutilizó gran parte de la progración Assembler correspondiente a modelos anteriores de Conmutadores.

Líneas por mes-hombre = 16700 / 41.8 = 399.5

A los efectos de este trabajo, es necesario comparar el tiempo invertido en programación auxiliar de Personalidades con el tiempo de desarrollo del LEP.

El replanteo especifico del LEP y de las Personalidades Dedicadas conduce a los siguientes resultados:

Medidas Directas:

	HORAS	MESES-HOMBRE
Diseño	410	2.4
Lexicográfico	90	0.5
Sintáctico	186	1.1
Semántico	153	0.9
Autómatas	334	2.0
Manuales y varios	80	0.5
TOTAL	1.253	7.4
Personalidades	690	4.0

Medidas Indirectas:

	PAGINAS	LINEAS	MESES-HOMBRE (estimados a 400 lineas/mes)
LEP	71	3.550	8.9
Autòmatas	7	350	0.9
TOTAL		3.900	9.8
Personalidades	44	2.200	5.5

Aûn sin entrar en otras consideraciones se puede decir que el tiempo de desarrollo de algunas personalidades dedicadas y el tiempo de desarrollo del LEP son comparables: el primero es el 60% del segundo.

Sin embargo, estos datos son relativos. Si se tiene en cuenta que sòlo se programaron personalidades dedicadas para tres Conmutadores de los cuatro instalados (las personalidades del cuarto fueron todas compiladas), surge que hubiera sido necesario un 80% del tiempo de desarrollo del LEP para obtener cuatro personalidades dedicadas.

Por otra parte sòlo se programaron personalidades dedicadas en el perìodo julio-diciembre, ya que en diciembre entrò en funcionamiento la primer personalidad compilada y a continuaciòn fueron sustituìdos los ejemplares dedicados por personalidades obtenidas mediante el LEP.

8. CONCLUSIONES

La experiencia acumulada en proyectos anteriores muestra que el mantenimiento de personalidades dedicadas implica tareas de reprogramaciòn, compilaciòn, linkediciòn y grabado de EPROMs durante los años de funcionamiento de un Conmutador. En el caso de los Conmutadores de 512 lìneas actualmente en funcionamiento, el mantenimiento y ajuste de Planes de Enrutamiento ha pasado a ser tarea exclusiva del personal de operaciòn.

Esto hace que al analizar los tiempos no con el criterio de trabajo inicial, sino con el criterio de desarrollo y seguimiento de una implementaciòn a lo largo de toda su vida ùtil, se pueda asegurar que la soluciòn LEP implica un enorme ahorro de recursos humanos y tècnicos que se suma a las ventajas de Operaciòn mencionadas anteriormente.

La aproximaciòn LEP representa realizar un esfuerzo de

abstracción, cuyo resultado es una definición precisa y de por sí orientadora en cuanto a la solución del problema. Este es el resultado esencial de la experiencia, mas allá de haber obtenido también una herramienta flexible y de gran utilidad en la Operación.

Sin pretender generalizar en exceso, los resultados analizados en el presente trabajo permiten considerar la aproximación LEP al menos como "no descartable a priori", en tanto solución de Ingeniería para problemas de un alto nivel de complejidad. Los resultados muestran, a través de un caso concreto, que la inversión de recursos en conceptualizar y abstraer los elementos principales de un problema es también una importante herramienta en el área de Ingeniería de Software.

8. BIBLIOGRAFIA

[1] "LA VIABILIDAD DE PROYECTOS NACIONALES: UNA CENTRAL TELEX CONSTRUIDA EN URUGUAY", J. Grompone, Boletín de Telecomunicaciones de la UIT, Vol. 49, No. V, páginas 280-284, Ginebra 1982.

[2] "COMMUNICATIONS TECHNOLOGY DEVELOPMENT: A CASE STUDY IN URUGUAY", J. Grompone, F. Simini, Communications Indonesia 1985 (The International Telecommunications and Business Communications Systems Exhibition), Jakarta, Indonesia 1985.

[3] "LENGUAJE HOMBRE-MAQUINA", Recomendación Z.311 a Z.341, Libro Amarillo, CCITT, Ginebra 1981.

[4] "THE THEORY OF PARSING, TRANSLATION AND COMPILING", Aho, Ullman, Prentice Hall, 1972.

PROGRAMAÇÃO LÓGICA E ENGENHARIA DE SOFTWARE

Teresa Cristina de Aguiar

COPPE - Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Caixa Postal 68511
21945 - Rio de Janeiro - RJ

RESUMO

Este artigo apresenta duas aplicações da programação lógica em Engenharia de Software: prototipagem rápida e sistemas especialistas que auxiliam no processo de desenvolvimento de software. Tem como objetivo discutir o potencial da lógica e sua adequação a estas duas aplicações.

1. INTRODUÇÃO

O interesse pela programação lógica tem crescido nos últimos anos devido principalmente ao fato da lógica simbólica estar sendo utilizada no projeto japonês de quinta geração. No entanto, antes disso, já se percebia que o formalismo da lógica poderia dar uma significativa contribuição à teoria e prática da computação. [HOGGER]

A diferença fundamental entre a programação lógica e a convencional é que a programação lógica requer a descrição da estrutura lógica de problemas, enquanto que na programação convencional há necessidade de se descrever como o computador deve agir para resolver um problema. Esta separação entre lógica e controle, proposta pela programação lógica, possibilita, segundo [KOWALSKI-79], que se alcancem alguns dos objetivos da Engenharia de Software: a construção de programas mais confiáveis, mais fáceis de serem manipulados e mais fáceis de se adaptarem a novos problemas. Para alcançar estes objetivos a Engenharia de Software tem se empenhado em melhorar o processo

de desenvolvimento de software, utilizando recursos como os da programação lógica.

Este artigo apresenta duas aplicações da programação lógica em Engenharia de Software: prototipagem rápida e sistemas especialistas que auxiliam no processo de desenvolvimento de software. Tem como objetivo discutir o potencial da lógica e sua adequação a estas duas aplicações.

Este artigo é parte de um trabalho que vem sendo realizado pela linha de pesquisa Engenharia de Software, da COPPE-UFRJ, com os seguintes objetivos: a) utilizar os recursos da Inteligência Artificial para auxiliar o processo de desenvolvimento de software, gerando, como consequência, produtos de alta qualidade, e b) dar o apoio da Engenharia de Software necessário à Inteligência Artificial.

2. APLICAÇÕES DA PROGRAMAÇÃO LÓGICA NA ENGENHARIA DE SOFTWARE

2.1- PROTOTIPAGEM RÁPIDA

Já não há dúvidas da necessidade de se descrever os requisitos lógicos de um sistema em sua primeira fase de desenvolvimento. Uma série de métodos têm sido criados com este objetivo. Quando a descrição dos requisitos é realizada numa linguagem lógica processável permite-se a análise e execução destes requisitos. Isso é de extrema importância, uma vez que não basta que na fase inicial de desenvolvimento de um sistema se descreva os requisitos - muitas vezes eles não expressam o desejado, assim como podem apresentar inconsistências. Esta capacidade de execução torna as linguagens lógicas ferramentas adequadas para construção de protótipos. Possibilita que antes

da fase de projeto e implementação se possa verificar se a especificação corresponde à visão que o usuário tem do sistema. Este tipo de verificação na fase inicial do desenvolvimento diminui bastante os custos do projeto.

A adoção desta estratégia implica na modificação do ciclo tradicional de desenvolvimento de software (especificação, codificação, testes, reformulação, recodificação e operação).

A nova estratégia propõe os seguintes passos:

1. Especificação inicial - É definida uma especificação parcial.

2. Testes interpretativos - A especificação (que ainda não é um programa) é testada através de interpretação, utilizando-se uma máquina de inferência.

3. Reformulação incremental - Se o comportamento mostra-se incompleto ou incorreto, a especificação é revisada e continua-se a partir do segundo passo.

4. Codificação - Se a especificação está correta, continua-se com o tradicional desenvolvimento de aplicações.

A codificação pode ser dispensada em certos casos, quando desejado, pois a análise executável tem a possibilidade de produzir diretamente o comportamento desejado para o sistema.

Considerando que a prototipagem rápida vem sendo desenvolvida com o uso de técnicas tradicionais, qual a vantagem de se utilizar a programação lógica nesta área? Os protótipos são geralmente elaborados na mesma linguagem de programação que é utilizada para a implementação. Empregando a programação lógica é possível implementar protótipos através de linguagens semanticamente equivalentes a diagramas de fluxo de dados [KOWALSKI-84]. [SIMPLICIO] descreve uma ferramenta capaz de

executar especificações de requisitos do sistema através da lógica.

2.2- SISTEMAS ESPECIALISTAS

A tecnologia de sistemas especialistas vem contribuindo para solucionar uma série de falhas que podem ocorrer com os especialistas de uma área: permite que o conhecimento, uma vez adquirido, não seja esquecido; facilita a transferência de conhecimentos; possibilita a geração de resultados mais consistentes, uma vez que o sistema especialista não está sob o efeito de fatores emocionais; a geração da documentação é facilitada, já que existe um mapeamento entre a forma que o conhecimento é representado no sistema e a descrição em linguagem natural desta representação. Os sistemas especialistas têm ainda a grande vantagem do baixo custo de operação, apesar do custo de desenvolvimento ser ainda bastante alto.

O desenvolvimento de sistemas especialistas de apoio à Engenharia de Software tem como objetivo armazenar o conhecimento de especialistas em algumas das etapas do processo de desenvolvimento de software, de forma a dar assistência aos especialistas ou prover informações ou auxílio àqueles que não têm acesso a um especialista da área. A lógica é uma das técnicas possíveis de serem utilizadas para representar o conhecimento de um sistema especialista.

Têm sido desenvolvidos sistemas especialistas em lógica com os seguintes objetivos:

- Possibilitar que programas sejam escritos numa linguagem de tal forma que permitam uma verificação e manutenção eficientes, além de utilizar de forma mais eficiente os recursos

que a linguagem de programação dispõe. O sistema especialista auxiliaria o usuário na programação e no uso mais eficiente da linguagem, dispensando-o de maiores treinamentos e da consulta a manuais. Um sistema especialista com este objetivo é apresentado em [ROMAGNOLI].

- Auxiliar o desenvolvimento da especificação de projeto, incorporando o conhecimento de um método para desenvolver este tipo de especificação. Um sistema com este objetivo é apresentado em [PESSOA].

3. LÓGICA E SUAS APLICAÇÕES

Segundo [BALDWIN] um sistema lógico tem vantagens importantes: a) o poder de expressão; b) as regras de inferência são corretas e completas, apesar de na prática muitos sistemas modelados não utilizarem o conjunto completo de regras de inferência e não exibirem todas as características de completeza.

No entanto o poder expressivo das fórmulas bem formadas e as características próprias das regras de inferência resultam em sérios problemas. Um deles é o longo tempo para construção de uma prova a partir de um objetivo e de premissas iniciais. Não existe um algoritmo para construir uma prova num tempo razoável, o que torna a lógica difícil de ser utilizada em grandes domínios. Outros problemas resultam do rigor na utilização das regras de inferência. Estas regras não permitem um raciocínio baseado em suposição ou por "default". O raciocínio baseado em probabilidade também não é permitido. É difícil, da mesma forma, utilizar a lógica em domínios nos quais proposições são verdadeiras somente num determinado momento. Para manipular proposições com um componente tempo, ou baseadas em probabilidade, há necessidade de se acrescentar mais fórmulas bem

formadas, aumentando-se o tempo de busca.

Apesar das limitações, as linguagens lógicas possuem um potencial que ainda não foi totalmente explorado. Há necessidade, portanto, de desenvolverem-se mecanismos que melhorem seu desempenho, o que já vem sendo realizado, uma vez que estão em fase de desenvolvimento uma série de trabalhos com este objetivo. Um deles é apresentado em [CASANOVA].

A partir da análise das aplicações apresentadas verifica-se que o gerador de protótipos é a que melhor justifica a utilização de uma linguagem lógica, uma vez que permite que se explore a semelhança entre a forma de se especificar e a forma de representação das linguagens lógicas. Além do fato de que, ao se utilizar uma linguagem processável, permite-se a imediata execução do que foi especificado, sem a necessidade de se utilizar outra linguagem, simplificando-se muito o problema de prova de correção.

Com relação ao desenvolvimento de sistemas especialistas não fica clara a vantagem da utilização da lógica. Existem outros métodos para o desenvolvimento desses sistemas e seria interessante se aqueles que optam pela lógica justificassem a escolha, analisando a possibilidade de utilização de outros métodos.

Percebe-se, no entanto, que de uma forma geral, a escolha das ferramentas utilizadas para o desenvolvimento de sistemas especialistas não tem sido realizada de maneira muito correta. Segundo [WATERMAN], muitos sistemas especialistas foram construídos com ferramentas escolhidas por motivos como: a) familiaridade do engenheiro do conhecimento com a ferramenta, b)

dentre as ferramentas compatíveis com o hardware do desenvolvedor, a escolhida era a mais eficiente; c) a ferramenta foi desenvolvida e depois foram escolhidas aplicações para testá-la.

Com relação ao desenvolvimento de sistemas especialistas, [KOWALSKI-83] afirma que PROLOG é uma excelente ferramenta de implementação. O que se pode dizer, independente de se utilizar uma linguagem lógica ou não para implementação, é que há certas dificuldades em se desenvolver sistemas especialistas. Primeiro, a dificuldade em se reunir um grupo de desenvolvimento competente, devido à falta de recursos financeiros. Segundo, as limitações atuais inerentes à tecnologia de Inteligência Artificial. E por fim o longo tempo necessário ao desenvolvimento.

Além destes problemas, um outro surge como uma séria dificuldade: a aquisição do conhecimento. Caso o método não seja descrito de uma maneira formal, a única forma de se adquirir o conhecimento sobre ele é através de consultas a um especialista, o que pode se tornar uma árdua tarefa.

É importante que antes de se iniciar a construção de um sistema especialista, seja realizada uma avaliação da viabilidade de sua construção. Segundo [WATERMAN], esta avaliação implica em se determinar não só se é possível construí-lo, mas também se é apropriado e justificado. Por não ser realizada esta avaliação têm sido desenvolvidos muitos sistemas que não podem ser chamados de especialistas, apesar de receberem esta denominação.

Com relação à aplicação de sistemas especialistas à Engenharia de Software, há ainda o problema do conhecimento em muitas áreas da Engenharia de Software não estar suficientemente

sedimentado, o que limita a construção de sistemas especialistas fora do ambiente acadêmico.

4. CONCLUSÃO

A lógica possui sérias restrições. No entanto estas limitações não impedem que as linguagens lógicas se tornem ferramentas eficientes em certos domínios de problemas, podendo tornar a computação mais eficiente. Quanto à Engenharia de Software, é uma área que como outras, pode vir a se beneficiar dessa nova tecnologia de software. Mas cabe também à Engenharia de Software o importante papel de fornecer, não só à área de linguagens lógicas como à Inteligência Artificial como um todo, recursos que aumentem a qualidade dos programas gerados e do processo de desenvolvimento de sistemas.

REFERÊNCIAS

[BALDWIN] D. Baldwin, G.M.Kasper; Toward Representing Management-domain Knowledge; Decision Suport Systems 2, 159-172; North-Holland; 1986.

[CASANOVA] M.A.Casanova, M.E.M.T.Walter; Implementação de Interpretadores PROLOG Combinando Expansão em Profundidade com Expansão em Amplitude; Anais do 3o Simpósio Brasileiro de Inteligência Artificial; Rio de Janeiro; Novembro 1986.

[DOYLE] Jon Doyle; Expert Systems and the "Mith" of Symbolic Reasoning; IEEE Transaction on Software Engineering; Vol. SE-11; Num. 11; Novembro 1985.

[HOGGER] C.J. Hogger; Introduction to Logic Programming; Academic

Press, Inc; 1984.

[KOWALSKI-79] R.Kowalski; Algorithm = Logic + Control; Communications of the ACM; Vol. 22; Num 7; Julho 1979.

[KOWALSKI-83] R.Kowalski; Logic Programming; Information Processing 83, R.E.A. Mason (ed); Elsevier Science Publishers B.V.; North-Holland; 1983.

[KOWALSKI-84] R. Kowalski; AI And Software Engineering; Datamation; vol. ?; 1984.

[MELLISH] W.F.Clocksin, C.S.Mellish; Programming in Prolog, Springer-Verlag; 1984.

[PESSOA] T.E.C.Pessoa, R.C.B.Martins, D.Shwabe; Um Sistema Especialista para o Método de Jackson; Anales de 1as 16as Jornadas Argentinas de Informática e Investigación Operativa; Buenos Aires; 1986.

[RICH] E. Rich; Artificial Intelligence; International Student Edition; Mc Graw-Hill Book Company; 1986.

[ROBINSON] J.A.Robinson; Logic Programming - Past, Present and Future; New Generation Computing; OHMSHA LTD; 1983.

[ROMAGNOLI] G.S.Romagnoli, M.T.M.P.Silva, M.C.Filho; Um Sistema Especialista para Programação em NATURAL; Anais do 3o Simpósio Brasileiro de Inteligência Artificial; Rio de Janeiro; Novembro 1986.

[SIMPLÍCIO] F.C.Simplício Filho; Um Método de Análise por Protótipos Baseado em Representação de Conhecimentos; Tese de Mestrado; PUC - RJ; Agosto 1986.

[WATERMAN] Donald A. Waterman; A Guide to Expert Systems, Addison-Wesley Publishing Company; 1985.

INGENIERIA DE SOFTWARE Y PROLOG

F. Javier Diaz

U.N.L.P.
y
E.S.L.A.I.
Cno. Gral. Belgrano Km 40
C.C. 3193
1000 Buenos Aires
Argentina

Resumen:

El presente trabajo discute las posibilidades de cumplir con los objetivos principales de la Ingenieria de Software utilizando el lenguaje Prolog. La idea central es que un buen lenguaje de programación para poder ser utilizado en toda su potencia necesita de un conjunto de herramientas que simplifiquen la tarea del diseñador.

Se describe en primer término el concepto "ambiente de programación", y posteriormente se lo analiza (con parámetros de la Ingenieria de Software) en función del lenguaje de programación Prolog.

Este trabajo se ha realizado en el marco del proyecto "Ambiente de Programación Inferencial" (API), financiado por el Programa Nacional de Informática y Electrónica de la SeCyT, Argentina (nro. 537-001).

I. Què significa "ambiente de programaciòn" ?

A medida que crece la complejidad de los problemas que se trata de resolver utilizando la computadora aparecen nuevas fuentes de dificultades para el desarrollo y la puesta a punto de los sistemas de procesamiento de datos. Usualmente se referencian estos problemas como "programming in the large" para diferenciarlos de la algoritmica tradicional, que se denomina "programming in the small" (GHE 87). Para atacar tales problemas se pensó que lo mejor era la utilizaciòn de principios de ingenieria para el dasarrollo de sistema complejos. De esta forma surgió la Ingenieria del Software. La misma trata principalmente los problemas vinculados con la disminuciòn de la complejidad de los problemas utiliando tècnicas de descomposiciòn y empleando metodologias que faciliten la elaboraciòn de las distintas partes, y su posterior integraciòn.

Resulta casi imposible pensar en atacar problemas reales de cierta magnitud sin hacer uso de ciertos principios fundamentales. Desde mediados de la dècada del 70 surgen los "ambientes para desarrollo de software" como una herramienta indispensable para englobar los distintos conceptos que conducen a la elaboraciòn y puesta a punto de sistemas (BAR 86). A continuaciòn se trata de caracterizar el concepto de "ambientes" de forma tal de tener parámetros concretos en funciòn de los cuales se pueda medir la calidad de los mismos.

I.1. Definiciòn tentativa de ambientes

Usualmente se entiende por "ambientes de programaciòn" un conjunto integrado de herramientas que posibilitan al programador cumplir con todas las etapas del ciclo de desarrollo de software utilizando un sistema de notaciòn homogèneo y sin necesidad de cambiar de contexto segùn la funciòn que desee realizar.

Usualmente en las etapas de desarrollo de software se empleaban herramientas desarrolladas ad-hoc y que, por tanto, respondian a una visiòn demasiado especifica del tema. Esto obligaba al diseñador de software a tener que manejar simultaneamente los objetivos (y lenguajes especificos) de quien desarrollò cada una de tales herramientas. De esta forma se obligaba al usuario a tener que realizar una serie de pasos, con frecuencia obvios, y aùn redundantes, los que distraian su atenciòn del objetivo final de su trabajo.

I.1.1. Definición general de "ambientes de programación".

Si bien es difícil dar una definición rigurosa que permita identificar, con total certeza, que es un ambiente de programación, existe concenso en el siguiente sentido, un sistema computarizado puede considerarse un ambiente de programación si reune las características siguientes:

I.1.1.1. Está compuesto por un conjunto de herramientas.

Esto significa que el sistema auna varias visiones distintas (cada una provista por alguna/s herramienta/s).

I.1.1.2. Se puede tener una visión integrada del mismo.

De este modo se permite una visión unificada del sistema fusionándose tareas que anteriormente eran consideradas por separado. Por otra parte se eliminan las jerarquías forzadas que dificultan la comprensión del sistema y obstaculizan su mantenimiento (por ejemplo haciendo más difícil el cambio de un cierto módulo).

I.1.1.3. Se simplifica su uso.

Se trata de que el sistema reaccione más inteligentemente, a fin de disminuir los requerimientos para ser usado efectivamente por el usuario. Esta característica es fundamental. La idea que predomina es la de evitar al usuario la codificación de una serie de pasos, cuando los mismos sean previsibles. Por ejemplo es claro que si se produjo un error de sintaxis en la compilación lo siguiente que hará el usuario es editar la línea donde se detectó el inconveniente.
Otro objetivo es simplificar la tarea del usuario, empleando una cantidad mínima de comandos -con una sintaxis cómoda-, a fin de hacer más breve el período de aprendizaje, y sencilla la comprensión de las tareas que realiza.

I.1.1.4. El nivel de interacción debe ser alto.

La necesidad de que el sistema "colabore" con el usuario en las tareas que este realiza, son el mejor indicador del tipo de comunicación que se requiere. Para comprender mejor este punto basta evaluar la gran aceptación de los sistemas basados en íconos, la aplicación de "mouse" o de "joystick", etc. De esta forma las capacidades del hombre y de la máquina pueden aunarse, sin que una forma de comunicación demasiado rígida y poco flexible obstaculize el proceso.

I.1.1.5. El sistema debe se completo.

No debe ser necesario salir del sistema para realizar alguna tarea que coadyude al propósito del mismo. Es

significa que el sistema debe prever todas las posibilidades. De esta forma se logra que con la utilización de un único medio se resuelvan todos los problemas asociados con determinado fin.

I.1.2. Ejemplos de componentes en un "ambiente de programación"

Para fijar ideas es conveniente repasar las componentes básicas para desarrollar software. Debe considerarse que si bien la enumeración dada parece adecuarse más bien a un enfoque clásico del desarrollo de software las actividades que las mismas cumplen están incluidas en los objetivos para la integración de un sistema complejo.

1) Editor de texto
2) Macroprocesadores o preprocesadores
3) Intérprete / compilador
4) Sistema de manejo de archivos
5) Editor de enlaces
6) Analizadores dinámicos de frecuencias
7) Optimizadores de programas fuente
8) Sistemas para depuración de programas/sistemas

I.1.3. Objetivos de los "ambientes de programación"

Por último mencionaremos los objetivos más generales a los que tienden los diseñadores de este tipo de sistemas.

I.1.3.1. Facilitar la tarea del programador

Se trata de proveer una herramienta lo más flexible posible para cubrir una gama de aplicaciones. No sólo se trata de hacer más simple la tarea (evitando pasos innecesarios) sino de brindarle al usuario una mayor gama de opciones de las que estaban disponibles hasta ese momento.

En este punto, es fundamental definir qué se entiende por "tarea del programador". Si bien, en general se asocia tal frase a todas las etapas de desarrollo de software, las mismas están condicionadas al modelo de ciclo de vida que se utilice. A continuación se da una lista de las tareas que se consideran de fundamental importancia:

i) **edición** (del/de los programas y documentación) Etapa de entrar la información en el sistema.

ii) **demostración de corrección**, verificación formal de que el programa responde a sus especificaciones.

iii) **depuración** o puesta a punto del programa, está vinculado a la adecuación del sistema realizado respecto de la realidad (o problema real con el que se enfrenta el usuario final).

iv) **documentación**, vinculado a tener facilidades tanto para mantener simultaneamente programas y texto explicativo de los mismos, como con la facilidad de deducir alguna documentación de los programas mismos.

v) **optimización** del sistema resultante, vinculada a la posibilidad de aplicar transformaciones al código que permitan mejorar la performance del sistema

vi) **mantenimiento de los sistemas**, si bien se la puede pensar como una síntesis de las otras tareas, tiene una problemática propia asociada a la facilidad de incorporar modificaciones (para adecuar al sistema a circunstancias cambiantes).

vii) **reusabilidad de código**, está relacionado con la posibilidad de adaptar las soluciones a diversos problemas que se resolvieron con anterioridad sin necesidad de una recodificación de los correspondientes fragmentos de código. Tal reusabilidad puede verse de dos ángulos, como la posibilidad de usar facilidades que brinda el sistema de soporte básico o poder re-utilizar fácilmente (y sin conflictos) código que anteriormente especificó el mismo usuario.

I.1.3.2. Incrementar la "inteligencia" del Sistema.
La tendencia actual es la programación auxiliada o asistida por computadora. Se pretende eliminar la tarea no creativa del programador. Por otra parte el sistema reacciona en forma variable según sea nivel de experiencia del usuario.

I.1.3.3. Disminuir la complejidad de los sistemas resultantes.
El objetivo es que las herramientas que estan ahora disponibles permiten intentar resolver problemas cuya complejidad los hacia intratables computacionalmente empleando los sistemas más clásicos.

I.1.3.4. Brindar un nexo más cordial con el usuario.
La característica saliente de estos sistemas es que su uso es agradable y mantienen una comunicación fluida utilisando la menor cantidad de convenciones (artificiales) que sea posible.

I.2. Objetivos del diseño de ambientes que afectan a las características de un lenguaje de programación.

La característica saliente de un ambiente de programación es que el lenguaje que debe emplear el usuario es homogéneo. Esto implica que debe incluir opciones para cubrir todas las necesidades que surjan. De este modo los lenguajes de programación más modernos incluyen opciones para la puesta punto, manejo de interrupciones, concurrencia, formas de encapsular módulos y de proveer una comunicación estandar. En particular los objetivos perseguidos pueden resumirse en los dos puntos siguientes.

I.2.1. Permitir su uso por no especialistas.
En particular si bien esto no puede cumplirse a extremo (ya que siempre es necesario algún grado de entrenamiento) la tendencia es que el lenguaje que se provee pueda definirse (y emplearse) por subconjuntos crecientes. Tales subconjuntos admiten una definición formal y cubren las necesidades de cierto tipo de aplicaciones. A medida que la complejidad de los proyectos que se pretenden resolver crece puede hacerse uso de herramientas más sofistificadas. Lo importante es que no fue necesario conocer las mismas hasta ese momento, y que al estar el lenguaje definido en forma incremental, la comprensión de la nueva estructura no afecta a lo que ya se conocia.

I.2.2. Brindar facilidades para el desarrollo de aplicaciones de gran envargadura.
Este punto es de fundamental importancia, por cuanto hace a que el lenguaje diseñado cumpla con los requerimientos de la Ingeniería de Software. Es bien conocido como cambia la forma de programar (desde el diseño al mantenimiento) de los sistemas de gran envargadura respecto de los estandares establecidos para la programación en pequeña escala (algoritmica). En este sentido pasan a ser prioritarios conceptos de seguridad. En particular surgen claramente el concepto de tipos (como forma de ocultar información y de evitar usos indebidos o no previstos de la misma) y el concepto de módulos (como forma de ocultar comportamiento y permitir la más fácil localización de los efectos laterales que ocasione el cambio de una parte del programa). Por otra parte, no es posible pensar en una ingeniería de sistemas de software sin poder reusar componentes anteriormente desarrolladas (pero

lo cual es imprescindible disponer de formas lo más general posible de implementar los conceptos arriba descriptos).

II. POSIBILIDADES DE PROLOG PARA DEFINIR UN AMBIENTE DE PROGRAMACION.

Es fundamental hacer un análisis de las características del lenguaje de soporte de un ambiente, con el fin de poder determinar hasta que punto es posible concretar los objetivos perseguidos. Con este porpòsito se enumeran a continuación los conceptos que deben estar presentes en un ambiente de desarrollo de software. En particular se analiza las posibilidades de Prolog (CLO 84, LLO 84, KOW 86), sobreentendièndose que, cuando no se lo mencione explícitamente, Prolog cumple con los objetivos enunciados.

II.1. Alto nivel de interacciòn.

Es claro que para proveer una interfase hombre/máquina razonable, es conveniente que el sistema subyacente (en el que se implementarà nuestro sistema) sea altamente interactivo. En particular esto permite que el sistema no sea estàtico y se pueda adaptar a circunstancias (y usuarios) cambiantes.

II.2. Sintaxis comùn de las primitivas del sistema y los predicados definidos por el usuario.

Esta caracteristica, que fue mencionada anteriormente cuando se habló de que el lenguaje fuese incremental, es fundamental si se trata de que el mismo sea homogéneo y simple.

II.3. Posibilidad de amalgamar datos y programas.

Esto significa que es posible regenerar el kernel de intèrprete, modificar programas y ampliar al sistema dinàmicamente. De esta forma el sistema podrà incorporar aquella informaciòn que considere necesaria y usarla como una extensiòn del ambiente predefinido. De esta forma, los programa entrados por el usuario, pueden ser transformados antes de ser manipulados por el sistema.

II.4. La existencia de una màquina de inferencias.

Prolog provee en su nùcleo una implementaciòn del mètodo de resoluciòn linear. De esta forma, el sistema siempre responderà segùn los axiomas lògicos que constituyen el programa. Es decir, no es necesario demostrar que el comportamiento del programa es el indicado en las especificaciones, sino que esto serà una consecuencia directa de trabajar en Prolog. (Sin embargo debe tenerse en cuenta que el sistema no valida la consistencia de los axiomas cuando se los

odifica, es decir pueden existir clausulas redundantes, y aun contradictorias).

II.5. Síntesis de programas.
Existen varios trabajos relativos a la posibilidad de sintetizar programas a partir de ejemplos y de derivar programas a partir de especificaciones en lógica.

II.6. Poseer una sintaxis natural.
La sintaxis de Prolog se asemeja a la de la lógica clausal de primer orden. Por su parte, la notación de la lógica matemática fue creada para poder modelizar con propiedad (eliminando ambiguedades) el lenguaje natural. De modo, que si bien difiere de este último, parece ser el formalismo que mejor se adapta a su representación. Por otra parte, es posible agregar módulos que realizan la conversión a Prolog de subconjuntos del lenguaje natural.

III. DESVENTAJAS DE PROLOG PURO desde el punto de vista de la Ingenieria de Software.

Ya se destacó anteriormente que un lenguaje de programación debe incorporar ciertos conceptos a fin de poder ser empleado con éxito en el desarrollo de sistemas de gran envergadura (BOB 85). Mas aún cuando la aplicación es crítica (es decir cuando las fallas no son tolerables) o el usuario final es quien usa libremente el sistema desarrollado, se hace imprescindible la incorporación al lenguaje de construcciones que permitan controlar tales situaciones (FRA 86).

III.1. Lenguaje chato.
No existen jerarquías en la definición de un programa. Si uno considera los lenguajes tradicionales, es común la existencia de bloques (o unidades de programa) que pueden anidarse y que proveen un medio para determinar el alcance de las variables (ya sean datos o nombres de rutinas).
Si bien puede considerarse que todas las claúsulas que definen determinado predicado constituyen la definición de un procedimiento (al cual se lo puede considerar como no-determinístico respecto de la acción que efectivamente realiza) se debe destacar que se puede referenciar a cualquier claúsula del programa. De esta forma no es posible, por ejemplo, definir un predicado padre que sólo sea accesible (o utilizable) desde el predicado hermano.

III.2. La estructura de datos si bien tienen un gran poder de representación no provee selectores built-in.

A pesar de que la existencia de tales selectores causaría que coexistan functores y funciones (ya que los selectores actuarían como tales) en la definición de los términos, existirían ventajas de expresividad (pues nos evitaríamos codificaciones engorrosas y aún la definición de predicados definidos ad-hoc)
Por otra parte también sería afectada la eficiencia en la manipulación de las estructuras de datos.

III.3. La no existencia de tipos.
Si bien la genericidad de los argumentos de los predicados provee una gran flexibilidad en el uso de los mismos. La falta de seguridades respecto del sistema de tipos hace que se puedan producir errores muy difíciles de detectar. En particular, tales seguridades son obvias si el usuario final puede ser una persona sin experiencia.
Algunos chequeos de tipos pueden realizarse si el programador tiene la disciplina de asociar distintos functores a datos de distinta naturaleza, y si la cabeza de cada predicado tiene previsto con qué functores hace matching. Sin embargo no parece aconsejable que todo el peso de utilizar tipos recaiga en el programador quien ve complicada su forma de codificación.
Por último, es factible pensar en extensiones del proceso de unificación, provisto por Prolog, que podrían incluir un chequeo de tipos.

III.4. No existen facilidades de debugging.
Considerado como un sistema de demostración de teoremas que simplemente busca inconsistencias, parece razonable que no incluya tales facilidades. Sin embargo, si consideramos Prolog como un Lenguaje de Programación (es decir un lenguaje en el cual se expresan modelos que se esperan reflejen el comportamiento de algún sistema dado) es claro que interesa saber el porque de su comportamiento, es decir saber si algún predicado esta mal definido, o si la secuencia de matchings que realiza no corresponde con lo que se deseaba expresar.

III.5. No existen facilidades para la modificación de los programas.
Por ejemplo cada vez que se corrige un programa de algún modo se está modificando la definición de un procedimiento. De este modo debería poder verse qué otras definiciones existen (de forma de estar seguro que la modificación introducida es compatible con las demás definiciones que forman un procedimiento). Tampoco existe la posibilidad de ver que cláusulas son las que lo invocan, a fin de ver si la modificación que se introduce puede tener algún efecto no deseado.

III.6. No existen variables globales
Las variables que se utilizan en la definición de cada cláusula son de efecto local a la misma. Esto se debe a que puede considerarse que cada predicado está afectado de tantos cuantificadores universales como variables libres posee

De esta forma si bien se favorece el principio de "localidad de efectos" y de "granularidad", es imposible usar variables no locales a una claùsula. Esto es particularmente molesto cuando en realidad un procedimiento està formado por todas las claùsulas que definen un predicado. De esta forma no se puede tener una variable que tenga alcance en todo un procedimiento. Por otra parte para simular el efecto de una variable global se deberia implementar un predicado cuyo argumento represente el valor corriente de esa variable (lo que implica que, para reflejar el comportaiento tradicional de una variable, se deberian usar caracteristicas extralògicas como agregar y borrar claùsulas)

III.7. No es posible especificar estrategias de busqueda alternativas

Si bien la programaciòn en lògica posee la caracteristica de evitar que el programador tenga que preocuparse del control, las implementaciones por una cuestiòn de eficiencia, implementan un mètodo de resoluciòn que no es completo. Es decir, el sistema puede no responder aùn cuando existan soluciones encontrables. De esta forma, parece ser conveniente permitirle que el programador especifique alguna forma de control. Una posibilidad seria indicar que para determinadas deducciones usarà una estrategia de control (distinta de depth-first-search) provista ex-profeso por el usuario.

IV. CONCLUSIONES

El presente trabajo muestra que existen posibilidades ciertas de definir extensiones al lenguaje de Programaciòn Prolog de forma tal de poder satisfacer los requerimientos de la Ingenieria de Software. Sin embargo las mismas deben respetar las caracteristicas de diseño del lenguaje, es decir, la cantidad de primitivas que se agregen debe ser minima y su introducciòn estar destinada a la solución de alguno de los problemas planteados en el punto III.

V. BIBLIOGRAFIA

BAR 86: BARSTOW, SHROBE & SANDEWALL: Interactive Programming Environments. Morgan Kaufman 86.

BRA 86: BRATKO: "Artificial Intelligence Programming in Prolog" Addison Wesley 86.

BOB 85: BOBROW: "If Prolog is the answer, what is the question" IEEE Computer, Sept 85.

CLO 84: CLOCKSIN & MELISH: Programming in Prolog. Springer Verlag 84.

GHE 86: GHEZZI & YAZAJERI: Programming Language Concepts John Wiley, 2nd ed, 1986.

KOW 86: KOWALSKI: "Logic Programming" Imperial College, marzo 1986.

LLO 84: LLOYD: "Foundations of Logic Programming" Springer Verlag 1984.

IMPLEMENTACION DE UN SISTEMA DE CONTROL INTERNO EN UN CENTRO DE PROCESAMIENTO DE DATOS

Carlos A. Petrella
Ministerio de Transporte y Obras Públicas
Montevideo - URUGUAY

RESUMEN:

El estudio plantea los principales problemas derivados del desarrollo de un Sistema de Control Interno en un Centro de Procesamiento de Datos, describiendo las distintas etapas que se cumplen para implantarlo.

Primero, se plantean problemas de capacitación del personal y las necesidades prácticas de ajustes organizativos; luego, el énfasis se traslada a los problemas derivados de la falta de continuidad del servicio, para proyectarse a continuación sobre el control de costos operativos y nivel de servicio y por último, hacer hincapié en la planificación como factor importante para impulsar el desarrollo de los sistemas de información.

La base de la estrategia propuesta para encarar la implantación del sistema tiene como objetivo crear una adecuada conciencia de la magnitud de los problemas derivados de la falta de control, buscando ganar la colaboración del personal en todos los niveles a través de procedimientos que establezcan objetivos comunes, identifiquen riesgos, definan áreas de responsabilidad individuales y utilicen estos resultados para medir la contribución de cada empleado al cumplimiento de los objetivos corporativos.

El estudio recoge planteos teóricos y experiencias prácticas de varios paises latinoamericanos, analizando propuestas, para solucionar los principales problemas que se enfrentan al desarrollar controles en los sistemas de procesamiento automático de datos.

PLANTEO DEL PROBLEMA

Uno de los principales problemas que enfrentaremos al desarrollar controles es el derivado de que en el pasado, los criterios de diseño de procedimientos de control sobre los sistemas informáticos han puesto énfasis sobre aspectos tecnológicos postergando la consideración de aspectos vinculados con la organización, su personal y la propia gestión de administración de los recursos. (Referencia: System Auditability and Control, (Control Practices Report) Preparado por The Institute of Internal Auditors, Inc.).

Es más, la definición de sistemas de procesamiento de datos (SPD) suele aplicarse en forma restringida limitándola exclusivamente a los equipos electrónicos y su software asociado. Sin embargo, un SPD incluye además otros componentes: personal, materiales y procedimientos que actúan sobre la materia prima (datos) para obtener el producto elaborado (información). Por supuesto, objetivamente no puede dudarse de la importancia de cualquiera de ellos, sin embargo frecuentemente se responsabiliza en forma genérica a la computadora de la mayoría de los incidentes que ocurren en torno a un SPD. Es más, quien no ha escuchado alguna vez "el computador se equivocó" sin que el acusador vislumbre sombras de sospechas de fallas en el entorno del mismo. Evidentemente los problemas de funcionamiento del sistema pueden deberse directa o indirectamente a cualquiera de sus componentes. A su vez estos componentes están muy interrelacionados, razón por la cual por ejemplo, una falla en los procedimientos repercute en la calidad del producto terminado tan fuertemente como un error de lógica en un programa, o un incidente en la operación del computador.

Así pues todo el sistema debe disponer de filtros de control de calidad en cada paso de la cadena de producción hasta la obtención del resultado deseado. El problema reside en que, a pesar de que existe concenso respecto a que los filtros son necesarios, los sistemas mecanizados suelen prescindir peligrosamente de ellos.

La desinformación lleva a diseñar sistemas sin los debidos controles y requisitos de auditoría que luego hacen muy dificultosa la evaluación de su funcionamiento.

¿Que pensaría un gerente bancario si se pagara un cheque sin control de saldos y sin autorización previa? Seguramente se alarmaría y realizaría una investigación del hecho tratando de determinar responsabilidades. Sin embargo como contrapartida,

¿qué controles se exigen sobre el acceso a los recursos, y la protección de registros, los procedimientos de recuperación y resguardo sobre los archivos de cuentas corrientes una vez mecanizados? Obviamente la indiferencia para evaluar el funcionamiento de un sistema mecanizado es solamente producto del desconocimiento de su impacto sobre la organización por parte de usuarios y especialistas.

Ante este panorama, queda claro que, debería haber siempre un vínculo estrecho entre las funciones de control interno y procesamiento de datos, no solamente a nivel de la organización de la unidad encargada de procesamiento, sino, a nivel de los sistemas de información que usan procesamiento electrónico de datos. Es más, los propios sistemas de información deberían ser considerados parte del activo de la organización y custodiados con tanto esmero como las mismas cuentas bancarias (Referencia: El control interno sobre un sistema automático de procesamiento de datos del autor).

LA IMPLANTACION DE UN SISTEMA DE CONTROL INTERNO

El enfoque planteado refuerza la idea de que la instalación de un sistema de control interno es un proceso dinámico que involucra a todos los integrantes de la organización y cuya cooperación es imprescindible para mejorar el comportamiento de los sistemas de información. Como consecuencia del caracter dinámico de los sistemas, el proceso de implantación de un sistema de control interno se va desarrollando ciclicamente y en cada ciclo se diferencian períodos: relevamiento, elaboración de una propuesta, implantación y evaluación.

Cada ciclo se inicia con un relevamiento para reconocer problemas que directa o indirectamente afectan la calidad del servicio. Luego se intentan aplicar estandares correctivos con el objetivo de normalizar el funcionamiento del área afectada. Posteriormente se definen mecanismos para controlar el proceso verificando los resultados, buscando eficacia primero, y eficiencia después. Por último sobre la base de la información recogida, se inicia un nuevo ciclo.

Una vez cumplido cada ciclo se retro-alimenta el sistema lográndose un más eficiente reconocimiento de problemas que provocan nuevos ajustes que mejorarán aun más el sistema.

A su vez como las necesidades de la organización van cambiando en cada ciclo (relevamiento, propuesta, implantación y evaluación) variarán los objetivos de acuerdo con su grado de evolución. En ese sentido, se podrían diferenciar etapas en la evolución en los sistemas de información y los Centros de

Procesamiento de Datos (CPD) que van desde la prácticamente inexistencia de controles hasta la madurez institucional y que aportan gradualmente garantías crecientes. La primera etapa, pone énfasis en la capacitación y los ajustes organizativos, pasando luego, en una segunda etapa, por los esfuerzos vinculados con el mantenimiento de la continuidad del servicio, para hacer después incapié en el nivel de servicio y el control de los costos operativos y proyectarse al final, en la cuarta etapa, sobre tareas de planificación, estructurando planes de desarrollo viables y no expresiones de deseo que no siempre pueden ponerse en práctica. Así pues, las etapas que se identifican en la evolución hacia la madurez son las siguientes:

1) La capacitación y los ajustes organizativos (búsqueda)
2) La continuidad del servicio (seguridad)
3) Los costos operativos y los niveles de servicio (control)
4) La planificación (desarrollo)

La definición de estas etapas no implica que por ejemplo, si el énfasis en un período está puesto en la continuidad del servicio no exista interés por abaratar costos de operación o no haya preocupación por la queja de los usuarios ante un nivel del servicio que consideran inadecuado.

A continuación, se detallan las características de cada etapa y las principales actividades que se derivan de las metas planteadas a cada nivel de la evolución.

ETAPA 1 BUSQUEDA

La primer etapa, al igual que al nacer un ser humano, es de descubrimientos, y apunta al conocimiento de sus propias posibilidades y la adaptación al ambiente.

Así, pues, el trabajo comienza por los ajustes en la organización. Cuando se busca realizar ajustes organizativos con vistas a mejorar los sistemas de control interno, la participación de todos los integrantes de la organización y en particular de los especialistas de su CPD es fundamental. El personal debe ser involucrado desde el comienzo del proceso de tal manera de que no sienta que los controles se le imponen sino que deben ser implantados para salvaguardar su propia fuente de trabajo y, más generalmente, los objetivos de la empresa protegiendo sus áreas críticas.

Una forma de involucrar al personal es enfrentarlo con los riesgos que cada uno individualmente asume al no disponerse de sistemas de protección y control.

A su vez, para que cada uno, en la medida de sus posibilidades, aporte soluciones a los problemas, debería brindarse un curso inicial sobre organización y control interno de los sistemas de información y más específicamente, para algunos de ellos, de la propia "cocina" de los sistemas: su CPD. Estos cursos deberían permitir introducir a los usuarios en el mundo informático y acercar a los especialistas al mundo empresarial para que de ambas vertientes se obtenga un diagnóstico integral de los puntos fuertes y débiles de la organización como un todo. Debe buscarse por esta vía una mayor integración para que todos reconozcan que el ambiente es el mismo y que las diferencias no son más que perspectivas personales al mirar el mismo terreno (Referencia: Introducción al procesamiento de datos del autor.).

La reorganización con un estilo abierto de participación permite eliminar brechas y descubrir el camino hacia la utilización de la computadora como una herramienta.

Con esta óptica, se podrían hacer ajustes organizativos sin limitar la motivación de las personas que están vinculadas directa o indirectamente con los sistemas, ya sea por haberlos desarrollado inicialmente o por ser sus usuarios más importantes. En esta primera etapa, deberían individualizarse las áreas claves de toda la organización y de su CPD en particular para concentrarse en ellas. Como producto derivado de este estudio, debería poder responderse a preguntas como:

¿Cuáles son los objetivos de esas áreas claves, incluyendo el CPD?

¿Qué limitaciones en los sistemas de control nos impiden o dificultan alcanzarlos?

¿Cómo puedo utilizar la organización y el personal del CPD para superar estas limitaciones?

¿Cómo puedo utilizar la computadora para ayudar a las personas a superar esas limitaciones?

Un primer resultado práctico de esta etapa es lograr que el personal se sienta parte activa y responsable de la implantación de cambios y un segundo resultado práctico es la generación de una lista de todos los recursos críticos de la instalación (hardware, software, materiales, personal y sistemas de información) con vistas a encarar la segunda etapa. La clave es entonces aportar conocimientos sobre el tema que con el adecuado incentivo se reflejan en ideas concretas que surjan naturalmente desde dentro sin que resulten impuestas. Esto, por supuesto, no implica que el trabajo del

asesor sea totalmente pasivo, sino que actua como un catalizador para acelerar las reacciones que buscan mejorar la organización.

Este es el principio para iniciar el cambio. Buscar IDEAS RENOVADORAS que, con el apoyo de la organización, se podrán convertir en proyectos y éstos luego en realidades.

ETAPA 2 SEGURIDAD

Obviamente, el ciclo de capacitación no termina en los conocimientos básicos impartidos inicialmente. Luego del incapié inicial en la identificación general de componentes en una unidad, el énfasis debe trasladarse a los aspectos vinculados a la CONTINUIDAD DEL SERVICIO. Esto se traduce, en primera instancia, en un análisis de los dispositivos de seguridad que tiene el sistema.

La organización debe evaluar los recursos y los incidentes potenciales para cuantificar los riesgos y hacer un dictamen de la situación. Para encarar este trabajo es sumamente práctico disponer de una lista general de control. (Referencia: 42 Suggestions for Improving Security in Data Processing Operations IBM)

Además, debe tenerse en cuenta que el estudio será más sencillo si se dispone de un inventario actualizado de recursos de la instalación. Así pues la tarea de realizar un inventario o bien actualizar uno existente es una piedra funadmaental para construir un sistema de control. (No se puede controlar lo que se desconoce). Ahora bien, esta tarea no debe realizarse a ultranza; incluyendo cualquier producto que se encuentre, ya que de ese inventario general, no todos los recursos tienen el mismo valor. Pensando en ello, pueden seleccionarse recursos sensitivos (aquellos cuya pérdida o destrucción causa un daño material importante a la organización (afectando incluso su situación patrimonial) tratarlos especialmente, como forma inicial de reducir la cantidad de elementos a considerar.

Ese tratamiento inicial incluye detalles de quien administra cada recurso sensitivo (PROPIETARIO), como lo hace (PROCEDIMIENTO), quien accede (USUARIO) y desde donde (LUGAR).

Además, es importante disponer de una historia de los resultados de la gestión hasta el presente. A partir de estos datos se verificarán los incidentes realmente ocurridos y aquellos que puedan ocurrir.
El problema es cómo determinar aquellos que potencialmente pueden ocurrir. Como posibles guías se pueden considerar las

exposiciones a las que se ven enfrentados genéricamente los servicios informáticos o más particularmente aquellos con marcadas semejanzas con el que está siendo ajustado.

Con esta óptica podemos considerar inicialmente la siguiente lista:

1) Falla en los equipos y los programas (destrucción de pistas en un disco, cintas magnéticoas averiadas,procesador fuera de servicio, errores de programación que dañan archivos).

2) Utilización maliciosa o criminal de los recursos (sabotaje, operación malintencionada del computador, espionaje industrial, extorsión).

3) Invasión de privacidad (obtención indebida de información con propósitos políticos, económicos o simple curiosidad).

4) Catástrofes (incendios, inundaciones, guerra, etc.)

La medida de la importancia de cada categoría para una organización en particular sólo puede darla la propia organización.

Difícilmente en un libro se encuentra una receta para encarar semejante tarea. Sin perjuicio de ello, no está demás estudiar los planteamientos hechos por otros, especialmente si existen variados ejemplos de situaciones reales que puedan asimilarse a las propias.

Este esfuerzo inicial debe idientificar aquellos incidentes que pueden ocurrir en la organización ya que nadie está prevenido de lo que desconoce.

(Referencia: Security Accuracy and Privacy in Computer Systems James Martin es el planteo inicial y ejemplos)

Una vez identificados los incidentes potenciales, deben analizarse los costos operativos de los mecanismos de prevención o protección y compararlos contra los riesgos derivados de los controles no implementados. Debe verificarse concretamente si el costo de la protección del recurso no es superior al costo de los percances derivados de la falta o insuficiencia de procedimientos de seguridad.

Para hacer la comparación, hay que estudiar el impacto que tienen los incidentes sobre el servicio, por ejemplo, imposibilidad de procesar, pérdida de información o bien invasión de privacidad por copia o acceso no autorizado.

Ahora bien, estos planteamientos no son más que herramientas auxiliares para determinar la real situación de una organización en caso general, pero volviendo al punto inicial, el problema de como encarar la seguridad en un sistema de procesamiento de datos, es más complejo que puntear una lista de posibles controles y sus ponderaciones relativas.

Una posible alternativa puede encontrarse en el desarrollo de un plan integral de revisión del aspecto seguridad en una instalación. Al respecto, el siguiente planteo en etapas podría ser una posible solución:

1) Definir la filosofía que aplicará la organización respecto de todos sus sistemas de protección.

2) Establecer el valor real que la organización dará a determinados recursos que deben protegerse.

3) Definir un presupuesto para seguridad.

4) Delimitar responsabilidades y definir procedimientos de control.

5) Poner en práctica los procedimientos.

La idea de este plan es que la dirección superior defina la escala de valores de la organización con respecto a los riesgos potenciales que se le plantean. Recién allí tendrá sentido el estudio técnico y a partir de él las recomendaciones que se efectúen.

Para lograr este objetivo debe enfrentarse al personal superior con las carencias que tiene su organización. Con ese objetivo, se puede hacer una encuesta de opinión para que quede determinada la situación del área bajo su supervisión en los aspecto vinculados al tema. Como paso siguiente y mediante entrevistas personales se puede enfrentar al gerente con los riesgos potenciales de los que él es responsable y que de acuerdo con la encuesta no ha asumido. Estas charlas pueden ser el punto de partida de una toma de posición gerencial ante el tema. (Referencia: The considerations of Physical Security in a Computer Environment IBM, ver Auditing the Security System).
Si el plan tiene éxito, definir objetivos es más fácil pues puede surgir del concenso de los interesados y tener un sentido global que alcance toda la instalación.

Esto último es importante pues debe tenerse en cuenta que los riesgos no pueden evaluarse independientemente. La protección

de un componente en desmedro de otro da la misma garantía que el eslabón más flojo de una cadena a la tensión que la misma puede soportar sin romperse. Por ejemplo una empresa que tiene sistemas de resguardo contra accesos no autorizados, incendios, cortes de energía, fallas prolongadas en los equipos pero resulta que se le inunda el centro de cómputos que está en el subsuelo por un problema de desagüe ante una lluvia muy intensa ¿se puede decir que es un ejemplo de perfección?

A partir de una definición adecuada de los recursos que deben protegerse y de una estimación de su valor, es mucho más sencillo distribuir la asignación de un presupuesto dado.

Obviamente, como en otros temas, el monto adjudicado al rubro seguridad es un problema de política de la empresa pero con la salvedad de que si se dispone de información pormenorizada de la situación de todos los bienes informáticos, se puede resolver el problema con mayor razón de causa.

Finalmente, ese estudio debe concluir con un informe sobre el nivel de seguridad actual y recomendaciones a corto, mediano y largo plazo, brindando estimaciones de costo-beneficio de la solución recomendada y sus alternativas. Si el plan tiene éxito, la implantación será más sencilla, pues contará con apoyo del nivel superior.

Obviamente, esto no garantiza su éxito final, resta "vender" la solución a todos los niveles para lo cual los detalles operativos deberán ser ajustados de manera que no se choque innecesariamente con el personal encargado de poner en práctica las medidas recomendadas.

ETAPA 3 CONTROL

El control del nivel de servicio y los costos operativos de una instalación surge como requerimiento adicional, una vez que existen ciertas garantías sobre la continuidad del servicio. Después de que las cosas "salen" es natural que se desee que "salgan bien" y a "costo mínimo".

La idea de un servicio "bien" prestado está vinculada con las expectativas del usuario respecto del mismo. Por tal motivo el nivel de servicio debe definirse en coordinación con los usuarios de los sistemas.

A su vez, estos niveles de servicio deben expresarse en términos tales que sea sencillo evaluar su cumplimiento. Algunos indicadores pueden ser los tiempos de respuesta máximos admitidos para una determinada transacción medidos en segundos,

los tiempos diarios de disponibilidad del servicio medidos en horas, las garantías de utilización medidos en porcentajes y con interrupciones máximas que serán admitidas medidas en unidades de tiempo.

Por ejemplo, para la transacción depósito en caja de ahorro, el tiempo de respuesta máximo será de 5 segundos. El servicio estará disponible durante todo el horario bancario y dos horas adicionales después del cierre. Podrá utilizarse respectivamente el servicio el 95% del tiempo de disponibilidad estimada y en ningún caso las interrupciones por problemas en casa matriz serán mayores de 3 horas. Para estos casos excepcionales de interrupción se dispondrá de un sistema local en cada dependencia y se bloquearán los movimientos cruzados entre dependencias.

Un buen punto de partida para establecer niveles de servicio es definir con precisión que utilizar para medir el servicio. Para ello es útil definir que se entiende por: disponibilidad, capacidad, rendimiento y confiabilidad.
Disponibilidad: es el tiempo en que el servicio está operativo.
Capacidad: es una medida de volumen de trabajo máximo que puede ser procesado en un tiempo fijo.
Rendimiento: es una medida de volumen de trabajo que es procesado en un tiempo fijo.
Confiabilidad: es la posibilidad de utilizar los recursos con la disponibilidad, capacidad y rendimiento establecidos.

Surge entonces una interrogante: ¿cómo medir esos factores en un sistema de procesamiento de datos?

Algunos indicadores para evaluar cada uno de los factores anteriores pueden ser;

DISPONIBILIDAD	Porcentajes absolutos
	Distribución en el tiempo
	Estadísticas de tipos de incidente
	Tiempos promedio de resolución de problemas
CAPACIDAD	Cantidades de instrucciones por segundo
	Velocidades de transferencia entre dispositivos de la configuración
	Velocidad en líneas de trasmisión
	Líneas por minuto
	Transacciones por segundo
	Trabajos por hora
	Traducciones por día
RENDIMIENTO	Tiempo de respuesta por transacción
	Porcentajes de utilización de los recursos
	Distribución de uso de los recursos

Programas modificados por unidad de tiempo/programador
Lineas de programas desarrollados por unidad de tiempo/programador

CONFIABILIDAD Encuestas
Información histórica

Además, la confiabilidad puede medirse sobre la base de indicadores que vinculen datos obtenidos indirectamente de DISPONIBILIDAD, CAPACIDAD y RENDIMIENTO, como ser:

- Disponibilidad durante una jornada completa
- Tiempo entre incidentes
- Duración de incidentes
- Distribución de los tiempos de respuesta en un lapso de tiempo dado.

Estos elementos pueden servir de referencia para cuantificar la situación de una instalación. A su vez, permiten obtener indicadores de utilización de los recursos.

El problema una vez identificados los niveles es cómo controlar su cumplimiento y su costo.

El eje es el control de actividades de producción y desarrollo y su comparación contra los estandares de la instalación o, más genéricamente, del mercado en ambientes similares. Un punto de partida para contralor de actividades es la obtención de reportes apropiados como subproducto de la bitácora de consola, de los sistemas de contabilidad automática o de herramientas de medición de performance.

Estos estudios permiten incluso estimar costos operativos por ambiente y más detalladamente por usuario llegando incluso a controlar la gestión de los especialistas dentro del CPD.

Adicionalmente, también es útil disponer de información de operaciones como ser: libro de incidentes conteniendo causa del incidente, fecha de apertura, fecha de cierre y solución adoptada; libro de quejas de los usuarios; libro de datos entrados y re-ortes distribuídos, conteniendo fecha, hora y descripción del envío; consumo de suministros (especialmente papel y cintas de impresora, para conformar resultados de calidad del servicio obtenido por otras vías (Referencia: Casos prácticos para estudio de Auditoría de Sistemas del autor).

Con respecto al área de desarrollo existen indicadores de productividad que se pueden extraer indirectamente del sistema de contabilidad aunque, hoy por hoy, es prácticamente

insustituible una metodología de desarrollo y control interno de aplicaciones que apunte a medir más la calidad que la cantidad de trabajo (Referencia: Programming Real-time Computer Systems, James Martin)

O ro aspecto importante es el vinculado con las tareas de mantenimiento de la instalación o, más genéricamente, con la determinación y resolución de problemas derivados de cambios.

La determinación de problemas es una especialidad; los procedimientos en el manejo de la determinación de problemas han dejado de ser un asunto que se pueda controlar empíricamente para dar lugar a un tratamiento científico, lo que exige una formulación teórica adecuada y muchos conocimientos té nicos (Referencia: El control de la gestión de cambios del autor).

ETAPA 4 PLANIFICACION

A medida que las organizaciones se van haciendo más grandes y complejas van perdiendo flexibilidad y deben soportar una mayor inercia ante los cambios. Con los vehículos sucede algo similar; es más fácil manejar un pequeño auto que un camión de 10 toneladas; a su vez, detenerlo, ponerlo en marcha o doblar implicará mayor o menor dificultad de acuerdo al peso y características de la carga. El problema no es solo la pericia del conductor (la dirección), también influye el estado de la carretera (ambiente), el acompañante (asesoramiento) y la hoja de ruta (planificación) en otros factores. Este último elemento es importante pues, sin una buena hoja de ruta no se puede preparar el vehículo con suficiente antelación para enfrentar los peligros de la carretera, lo que aumenta el riesgo de accidentes a la vez que obliga a reducir el régimen de marcha.

¿Cómo combatir los problemas derivados de la administración de organizaciones muy grandes?
Una respuesta, sin duda, es: Planificar a largo plazo. Ahora bien, inmediatamente surge la pregunta.
¿Cómo planificar a largo plazo?
Primero hay que tener definidas políticas a nivel superior, en base a estas políticas establecer objetivos y en concordancia con ellos, planes a mediano plazo.
Es como levantar un edificio en el que en cada piso se asienta en el inferior y todos ellos en los cimientos con la diferencia de que se lo contruye de arriba para abajo.

Incluso hay quienes afirman que todavía hay un nivel más alto, los "dogmas" que le dan continuidad a una organización por generaciones (Referencia: Una Empresa y sus Credos, WATSON Jr.).

El problema es como construir ese edificio.

El principio es el mismo que en cualquier actividad: involucrar el nivel superior, a aquel que dirige y toma las decisiones más importantes. Con este criterio, el primer reto es integrar a la Administración Superior en grupos de trabajo interdiciplinarios para la definición de los grandes sistemas de información que apoyan a la organización para alcanzar las metas políticas que ésta se ha propuesto.

Habrá que trabajar luego en la definición práctica de los objetivos fundamentales de cada área funcional, identificando problemas de carácter general y definiendo planes concretos para corregirlos. Recién en este punto se deberá pensar en los problemas técnicos derivados de la mecanización de los sistemas y los recursos para emprender su implementación por computadora. Luego los recursos financieros definirán la estrategia o seguir para la mecanización de los sistemas.

Sin embargo, este punto es crítico ya que en esta etapa pueden perderse de vista los objetivos de la organización al ser "sustituídos" por los objetivos del propio CPD que suelen estar en manos de especialistas sin que se arbitre ningún sistema de control que rectifique posibles desviaciones.

Por sobre todo no debe olvidarse que la informática es un recurso que, aunque muy versátil, no tiene un fin por si mismo sino que es un medio para operar, controlar y decidir en una organización.

Una herramienta sumamente útil para controlar la gestión y para asegurar que los planes se ajusten a las necesidades de la organizacion y que puestos en marcha se cumplan es la DIRECCION POR OBJETIVOS (DIPO) que aplicado al área informática garantiza como beneficio adicional una visión más global de los problemas de una empresa.

Además, el ciclo de la DIPO ayuda a controlar la gestión ya que no solamente permite detectar áreas claves definiendo objetivos para cada uno de ellas sino que orienta para establecer planes de acción concordantes con dichos objetivos y "obliga" a establecer puntos de control para evaluar resultados y retroalimentar el sistema evitando grandes desviaciones (Referencia: Curso de Introducción a las técnicas DIPO de J. Antonelli).

CONCLUSIONES

Los controles deben abarcar todos los componentes del sistema. Esto es personal, equipos, materiales y procedimientos que son

utilizados en el ciclo de elaboración de la información. A su vez, debe existir cierto equilibrio que no deje abiertos flancos de mayor exposición por haber concentrado los controles sobre un área en desmedro de otra. En la medida en que los controles deben ser aceptados como soluciones a los problemas planteados, su introducción debería surgir naturalmente y ser apoyada desde el seno de la organización. Para lograrlo de la misma manera que un bebé comienza gateando para luego caminar a los tropezones y, por último, aprender a caminar correctamente e incluso a correr, la implementación de sistemas de control interno requiere un proceso de concepción y desarrollo para luego alcanzar la madurez y por que no, en algunos casos acogerse a la "jubilación" cuando ya no esten en condiciones de rendir los frutos que en otra etapa brindaron.

Los problemas más difíciles de resolver al implantar un sistema de control interno, no son los estrictamente técnicos. El talón de Aquiles en la estrategia de implantación es ganar la colaboración del personal. Debe buscarse entonces un procedimiento por el cual el personal de Dirección y sus subordinados identifiquen objetivos comunes, definan áreas de responsabilidades de cada persona de acuerdo con los resultados que se esperan de su trabajo y utilicen esos resultados para medir la contribución de cada uno de sus empleados al cumplimiento de los objetivos corporativos.

De esta manera, podrá encaminarse la fuerza individual de cada persona, será más fácil establecer formas de trabajo grupales y podrá armonizarse las metas individuales con el fin común.

No debe olvidarse que la real dificultad al impulsar nuevos sistemas de control se encuentra en el espacio interior, ese que todo hombre tiene entre oreja y oreja y que requiere tiempo para ser modificado. Además, en ese espacio es difícil imponer cambios; hay que lograr que sean aceptados y para ello hay que estar dispuestos a actuar con flexibilidad para luego recoger los frutos del esfuerzo.

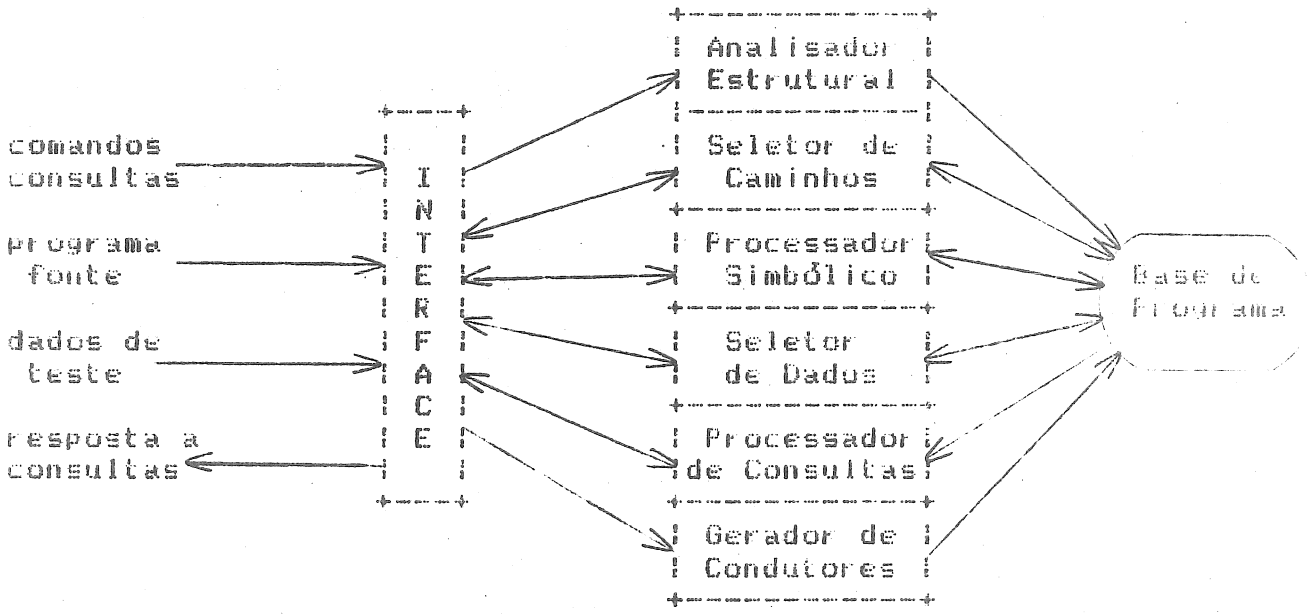
PROTESTE - UNA FERRAMENTA DE APOIO AO TESTE DE PROGRAMAS

ANA M. PRICE

Curso de Pós-Graduaçao em Ciencia de Computaçao
Universidade Federal do Rio Grande do Sul, Brasil

Descrevem-se as principais funções de uma ferramenta de apoio ao teste de programas que está sendo desenvolvida no Curso de Pós-Graduação em Ciência da Computação da UFRGS. Dentre as facilidades a serem fornecidas pela ferramenta proposta, encontram-se:

- análise estática de programa;
- assistência a seleção de dados para teste;
- geração de módulos condutores de teste;
- atendimento à consultas do programador; e
- documentação do plano de teste do programa.



A ferramenta aceita, básicamente, três classes distintas de dados de entrada:

- comandos/consultas do usuário, que incluem por exemplo, pedido de análise de programa, execução simbólica de um dado caminho de teste, projeção do grafo de fluxo do programa, etc.; os comandos serão apresentados através de menus;
- código fonte do programa a ser analisado, gravado em disco; e
- dados de teste, que serão introduzidos quando da interação do usuário com o módulo de seleção de dados.

As respostas às consultas serão mostradas através de janelas distintas, projetadas simultâneamente na tela.

Os módulos principais que compõem o corpo da ferramenta são: - Analisador Estrutural, que faz uma análise estática do programa produzindo uma tabela de símbolos e referências, um grafo dirigido representando o fluxo de controle do programa e um código intermediário interpretável;

- Seletor de Caminhos, que baseado no grafo de controle produzido pelo analisador estrutural e no critério de cobertura de teste escolhido pelo programador, produz uma enumeração dos caminhos do programa a serem testados;

- Processador Simbolico, que interpreta simbólicamente os caminhos préviamente selecionados; o resultado desta etapa é um conjunto de predicados, sendo cada predicado correspondente a um caminho de teste;

- Seletor de Dados, que partindo de um predicado de caminho tenta determinar um subconjunto do dominio de dados de entrada cujos elementos causam a execução do caminho considerado. O seletor de dados, entao, verifica e registra os dados de teste sugeridos pelo usuário para o caminho de teste em questão;

- Gerador de Condutores ("drivers" e "stubs"), que gera automaticamente esqueletos de programas, os quais servirão de suporte, tanto para o teste individual de módulos (para substituir subrotinas chamadas, mas ainda não testadas) como módulos de controle durante o teste de integração; e, finalmente,

- Processador de Consultas, que visa atender a consultas do usuário, tais como, "projete o grafo de controle do programa", "mostre o código fonte associado ao caminho de teste 1-2-4-7-8", "diga onde foi referenciada a variável X no caminho 1-2-4-7-8", "imprima o plano de teste do programa P", etc.

Com exceção do módulo Processador de Consultas, que apenas recupera informações sobre o programa, todos os outros módulos do sistema contribuem com dados para a Base de Programa. Dentre as informações armazenadas na Base, encontram-se: a tabela de símbolos e referências, o código intermediário interpretável, o grafo de controle de fluxo, caminhos de teste e predicados associados, dados de teste e resultados esperados por caminho, e dados complementares para condutores de teste.

PROTESTE está sendo desenvolvido para processar programas escritos em Pascal. No entanto, o sistema poderá ser extendido para as linguagens estruturadas em geral, pois os módulos do sistema, com exceção do Analisador Estrutural, trabalham sobre as representações internas do programa, independentemente do código fonte. O método estrutural de teste foi escolhido para servir de base à ferramenta, pois inclui diretivas para a seleção de dados de teste e pode ser razoavelmente automatizado. O estado atual de desenvolvimento do projeto compreende uma implementação parcial do Analisador Estrutural, desenvolvimento de um programa gerador de grafos de fluxo de controle na tela, e projeto do processador simbólico.

LAS REDES PETRI COMO UN INSTRUMENTO PARA MODELAR DIAGRAMAS DE FLUJOS DE DATOS JERARQUICOS.

Ricardo Acevedo, Depto. Informática,Universidad Santa María.Casilla 110-V
Valparaíso-Chile.

RESUMEN.

Una preocupación básica en el ámbito del análisis de sistemas, lo constituye la búsqueda de herramientas que permiten modelar los diagramas de flujo de datos con que se representan los sistemas de información.

En viturd de ello se presentará una técnica de modelaje conocida como "redes Petri", adaptada para su uso en la confección de diagramas de flujos de datos. Esta técnica fue concebida como un instrumento para representar sistemas de lógica dinámica. Orientada más bien a la teoría de autómatas y sistemas digitales, es posible de utilizar en cualquier sistema como actividades explícitamente estructuradas dadas sus facilidades para representar actividades alternativas, paralela, cíclica y aquellas con coordinación asincrónica en una perspectiva jerárquica y recursiva.

En esencia se conforman en base a tres componentes: estados, eventos y las relaciones secuenciales entre los estados y los eventos, en donde un estado que es finalizado por un evento se denomina "estado input" para ese evento y un estado que es iniciado por un evento se llama "estado out put" para ese evento.

En consecuencia, se presentarán las bases conceptuales, los elementos que las definen y el modo de utilizarlas como instrumento para modelar las representaciones de sistemas de información conocidas como diagramas de flujos de datos.

Proyecto MIS

METODOLOGIAS INTEGRADAS DE SISTEMATIZACION

Autor: Mgr. Jorge Eliécer Barrera Niño.
Jefe de Seguridad de los Datos, BAVARIA, S.A.
Profesor de Cátedra de la Universidad Nacional de Colombia.

Dirección: a. Cra. 10 No. 28-49. Piso 18 (Torre A de Bavaria)
b. Apartado Aéreo 249912
Bogotá, D.E., República de Colombia

El Proyecto MIS, de enfoque académico/práctico, integra las más modernas metodologías de sistematización en las áreas de :

1. Planeación de los Datos, en términos de la Ingeniería de la Información y de la metodología Business System Planning (BSP).

2. Proyectos de Sistematización, que comprende una metodología para desarrollo de sistemas, los estándares de meta-datos y su empleo dentro de la metodología y el manejo de proyectos.

3. Directorio de Datos, a la que corresponden los aspectos del Modelo Entidad-Asociación, del Modelo Normalizado de los Datos, del Diseño Físico Tanto tradicional como de bases de datos y de la distribución de los Datos.

4. Sistemas de Información que cubre el potencial de procesamiento de datos de los sistemas, su estructura lógica y la categorización de sus servicios según los diferentes niveles de usuarios.

5. Administración de Sistemas, que comprende el Manual de Procedimientos, el manejo del ambiente de producción de los sistemas, el control de acceso a archivos y bibliotecas y el soporte para la generación de planes de contingencia.

La integración de las mencionadas áreas, basada en un modelo general normalizado de meta-datos y automatizada en modalidad interactiva, generará una Metodología de Sistematización sin precedentes, desde el punto de vista de la coordinación lograda entre esas áreas.

La estrategia de desarrollo del proyecto considera entre otras cosas el empleo de lenguajes de muy alto nivel y el descarte de productos comerciales de software relacionados con cualquiera de sus áreas, en razón de sus no satisfactorias integración, cobertura y portabilidad, además de lo elevado de sus costos.

Dentro del marco conceptual del proyecto MIS, basado en los resultados de varias tesis de grado afines realizadas en la Facultad de Ingeniería de la Universidad Nacional de Colombia y en el aprovechamiento de experiencias relacionadas propias del medio y externas, se desarrolla inicialmente en la División de Sistemas de Bavaria, S.A. el soporte automatizado de los items concernientes a la Documentación de Sistemas y Archivos en Producción, áreas éstas estrechamente relacionadas con el manejo de la seguridad de los datos.

DISEÑO Y MANUFACTURA ASISTIDOS POR COMPUTADOR

MGIT-ANDES. UN MODELADOR 3-D SOBRE MICRO-COMPUTADORES, BASADO EN OPERADORES DE EULER.

JOSE TIBERIO HERNANDEZ P*.
JUAN C. QUIROGA B*.
Universidad de los Andes, Bogotá D.E.

En el artículo es estudiado el problema de modelaje de sólidos rígidos, introduciendo los aspectos teóricos y presentando las características principales del modelador MGIT-ANDES. Varios esquemas de representación de sólidos han sido desarrollados, destacándose los enfoques CSG (Constructive Solid Geometry) y B-Rep (Boundary Representation). Este modelador se basa en una variante del esquema de representación B-Rep y utiliza los operadores de Euler para mantener la validez de las representaciones. El proceso de creación se realiza en forma interactiva con la técnica de barrido (barrido por traslación y barrido por rotación). Las operaciones espaciales de conjunto: unión, intersección y diferencia, se han implementado como ayuda para la definición en un espacio completo de sólidos tri-dimensionales.

Palabras Claves : Representación de la frontera (B-Rep), Operadores de Euler, Definición por Barrido, Operaciones de Conjunto.

1. INTRODUCCION

Un modelador geométrico es la parte de un sistema DAC (Diseño Apoyado por Computador) que permite representar y manipular

* Este trabajo es producido por el grupo DFAC de la facultad de Ingeniería, departamento de Sistemas y Computación de la Universidad de los Andes, del cual los autores son miembros. No se debe hacer copias del mismo sin la respectiva autorización.

espacialmente objetos tri-dimensionales. Los cálculos generales sobre el objeto definido (volumen, etc.) son realizados por el sistema modelador. Con esto, el esquema de representación no se limita al dibujo del objeto modelado. La representación que se crea de un objeto, para garantizar el cumplimiento de estos requerimientos, debe designar en forma no ambigua ese objeto y conservar la información necesaria (topológica y geométrica) para efectuar los cálculos sobre el modelo.

En este artículo se describe el desarrollo del modelador geométrico MGIT-ANDES y se presentan sus principales características. Inicialmente se introducen los fundamentos teóricos del esquema de representación adoptado. Luego se describe el diseño para el modelador referido.

2. MARCO TEORICO

Para garantizar la corrección en las operaciones de manipulación espacial y de cálculos generales sobre los elementos físicos, los objetos físicos deben ser *representados* en forma tal que los modelos resultantes los designen en forma no ambigua y contengan los datos suficientes para realizar los cálculos requeridos. Han sido desarrollados varios esquemas de representación de sólidos rígidos para modeladores geométricos [Brai 1975] [Fumi 1984] [Requ 1980] [Requ 1982]. Se destacan, por el dominio que se tiene de ellos, el esquema de representación CSG (Constructive Solid Geometry) [Requ 1977] y el de representación B-Rep (Boundary Representation). Este último esquema es de nuestro interés y se enuncia a continuación. Su unión con los llamados *Operadores de Euler* permite ver el problema de representación de sólidos rígidos como la definición de un tipo abstracto de datos, en el sentido de [Gutt 1977].

B-Rep. Representación de la frontera.

Un sólido es definido por una colección de caras, aristas y vértices y sus relaciones espaciales [East 1979] [Mart 1982]. Las operaciones de definición en sistemas B-rep aseguran que la frontera está bien formada: no se intersecta ella misma, es cerrada y orientable [East 1979].

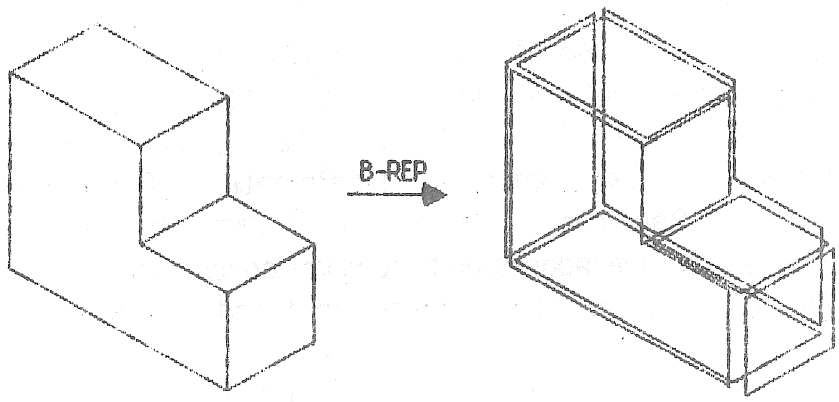


Figura 1.

Un modelo por la frontera puede ser representado simplemente como un conjunto de polígonos interconectados, los cuales a su vez se representan por su frontera por medio de *loops*. Se define un loop como un polígono cerrado que forma, junto con otros, los límites de una *cara* Un loop define el límite exterior, mientras que otros describen las cavidades. Estos datos, caras, loops y aristas describen la topología del modelo, mientras que los datos tales como ecuaciones de las caras o coordenadas de los vértices describen la geometría del modelo [Wilf 1985]

Para lograr mejores desempeños en las operaciones sobre el modelo, se incluyen en la representación otros datos explícitos de las relaciones entre caras, aristas y vértices del objeto.

Operadores de Euler.

No todo conjunto de caras define un sólido físico válido. La integridad topológica de un modelo B-rep incluye restricciones en los conjuntos de caras, de manera que se asegure la validez de las representaciones [East 1979]: cada arista debe pertenecer exactamente a dos loops (no necesariamente distintos), el orden de las aristas en los loops debe ser consistente en todo el modelo (por ejemplo en el sentido de giro de las manecillas del reloj), las caras se pueden intersectar solo en las aristas o vértices comunes y, finalmente, la frontera del sólido debe ser conexa. Baumgart [Baum 1972] demostró que basando el modelo B-Rep en los operadores de bajo nivel conocidos como Operadores de Euler, se garantizan las condiciones mencionadas arriba.

Estos operadores manejan las estructuras de datos en el modelo B-rep y garantizan que las representaciones sean válidas. Derivan su nombre de la ley de Euler, la cual establece que en un poliedro simple el número de caras (f), aristas (e) y vértices (v) deben satisfacer la ecuación:

$$v - e + f = 2$$

La fórmula puede ser generalizada para cualquier sólido introduciendo el número de anillos (cavidades en las caras, r) y el número total de huecos (h) a través del sólido. En general, la fórmula es:

$$v - e + f = 2 (1 - h) + r$$

Los operadores de Euler trabajan sobre la topología de un modelo

B-rep, esto es, sobre las relaciones espaciales de sus caras, aristas y vértices. Sólamente cinco operadores (con sus inversas) son suficientes para describir todos los objetos que satisfacen la fórmula de Euler. El siguiente grupo ha sido tomado como ejemplo de [Mart 1982].

Operador	Descripción
mvsf(f,v)	CREE VERTICE, SOLIDO, CARA
kvsf( )	DESTRUYA VERTICE, SOLIDO, CARA
mev(v_1,v_2,e)	CREE ARISTA, VERTICE
kev(e,v)	DESTRUYA ARISTA, VERTICE
mef(v_1,v_2,f_1,f_2,e)	CREE ARISTA, CARA
kef(e)	DESTRUYA ARISTA, CARA
kemr(e)	DESTRUYA ARISTA, CREE ANILLO
mekr(v_1,v_2,e)	CREE ARISTA, DESTRUYA ANILLO
kfmrh(f_1,f_2)	DESTRUYA CARA, CREE ANILLO, HUECO
mfkrh(f_1,f_2)	CREE CARA, DESTRUYA ANILLO, HUECO
semv(e_1,v,e_2)	DIVIDA ARISTA, CREE VERTICE
jekv(e_1,e2)	UNA ARISTAS, DESTRUYA VERTICE

3. MGIT-ANDES : UN MODELADOR DE SOLIDOS.

MGIT-ANDES es el nombre del modelador de sólidos experimental desarrollado en la Universidad de los Andes [Brav 1985] [Quir 1986]. Pertenece a la clase de modeladores de la frontera, basado en operadores de Euler. Las técnicas de *barrido por traslación y rotación* se utilizan en la creación de modelos de sólidos junto con las *operaciones de conjunto*, alcanzando en esta forma un grado natural y eficaz de definición. Incluye manipulaciones espaciales de traslacion, rotación y escalamiento sobre los modelos definidos. Con la interfase para otros programas, el

modelador se integra a un sistema CAD facilitando la visualización de los pasos intermedios en los procesos automatizados.

Siguiendo los principios de modelaje de la frontera, se adoptó un esquema de representación en el que las caras son planas y las aristas segmentos de recta, cuyo dominio está constituido por la familia de todos los poliedros de caras planas. Este esquema da lugar a la siguiente estructura para la representación de un sólido :

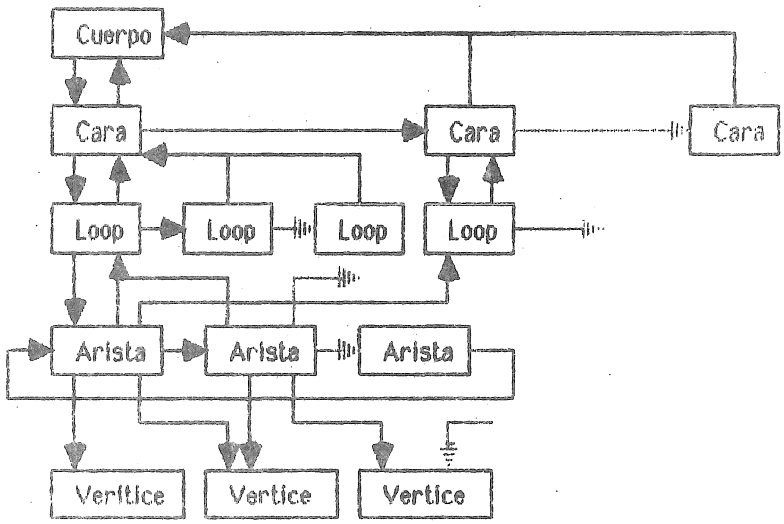


Figura 2.

Esta estructura contiene tanto los datos topológicos como geométricos del objeto. Cada *vertice* contiene las coordenadas correspondientes a un vértice del objeto. Cada *arista* se relaciona con sus *vertices*, con los dos *loops* a que pertenece y con la *arista* siguiente en cada uno de estos *loops*. Un *loop* se relaciona con la *cara* a que pertenece, con el siguiente *loop* en esa *cara* y con el conjunto de *aristas* que lo definen. Una *cara* se relaciona con el *cuerpo* a que

pertenece y con los *loops* que la describen. Un *cuerpo*, finalmente, está descrito por un conjunto de *caras*.

Para construir y mantener esta estructura de datos, se implementó el siguiente subconjunto de operadores de Euler. Mäntylä [Mart 1984] demuestra cómo estos son suficientes para representar cualquier objeto.

Operadores constructores	Descripción
Cvcc	CREA VERTICE, CARA, CUERPO
Cav	CREA ARISTA, VERTICE
Cac	CREA ARISTA, CARA
CcDah	CREA CARA, DESTRUYE ARISTA, HUECO
MeKr	CREE ARISTA, DESTRUYA ANILLO
Glue	UNA DOS CARAS

Operadores simplificadores	Descripción
Dvcc	DESTRUYE VERTICE, CARA, CUERPO
Dav	DESTRUYE ARISTA, VERTICE
Dac	DESTRUYE ARISTA, CARA
DcCah	DESTRUYE CARA, CREA ARISTA, HUECO
SeMv	DIVIDA ARISTA, CREE VERTICE

Creación Interactiva de Sólidos

La interfase con el usuario debe permitir una manipulación facil de los elementos durante el proceso interactivo de definición. La técnica de barrido (por rotación y por traslación) y las operaciones espaciales de conjunto (unión, intersección y diferencia) logran este objetivo gracias a un conjunto de herramientas. En el modelador, las herramientas de manipulación permiten trasladar, rotar, escalar, posicionamiento de una cara y visualización con linea escondida, en cualquier momento del proceso de creación.

Definición por barrido.

Utiliza el elemento *cara* como elemento generador de la representación. El sólido se define por el conjunto de puntos comprendidos por la cara generadora y una trayectoria determinada. Los operadores de Euler intervienen validando la trayectoria y actualizando la estructura de datos. Sobre el elemento generador se pueden efectuar operaciones de escalamiento, así como definir nuevos loops interiores. Esto último permite definir tanto salientes, como huecos en el modelo. El modelador brinda las dos formas usuales de barrido : por traslación y por rotación, las cuales pueden usarse en cualquier momento en el proceso de creación de un sólido.

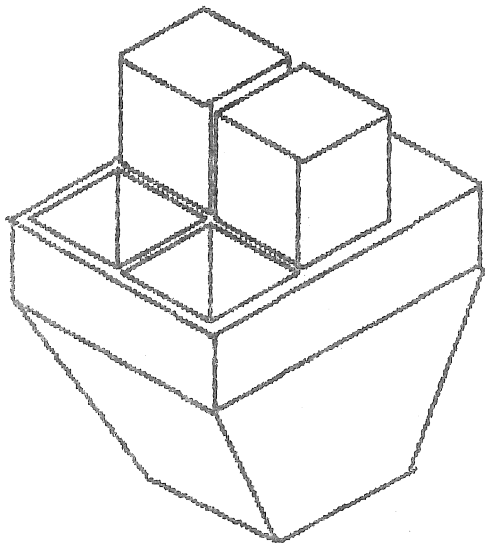


Figura 3.

El conjunto de sólidos que se puede modelar en esta forma es limitado, aunque no trivial. Con la implementación de las *operaciones espaciales de conjunto*, se obtiene una forma natural de definición para un conjunto completo de sólidos.

Definición con Operaciones de Conjunto.

Con las *operaciones de conjunto* se realizan las operaciones unión, intersección y diferencia sobre los modelos definidos. Aunque presentan una interfase natural para la creación, sus algoritmos suelen ser bastante complicados y son tema actual de investigación.

Para la implementación de algoritmos de operaciones de conjunto para objetos de caras planas existen varios enfoques [Chiy 1985], [Mart 1983]. El diseño inicial de MGIT-ANDES se basó en las ideas expuestas por Mäntylä y Tamminen, por tanto la implementación de la operaciones de conjunto sigue el enfoque de [Mart 1983]. A grandes rasgos, en la ejecución de una operación de conjunto primero se determinan las caras que forman los sólidos resultado y luego se "unen" usando los operadores de Euler.

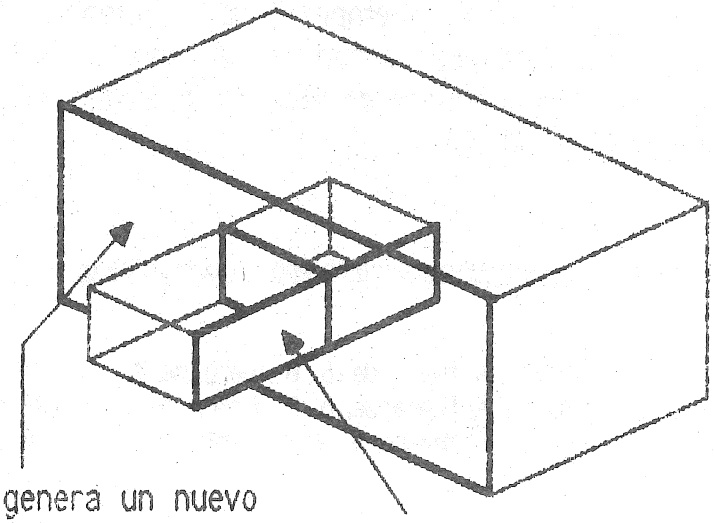


Figura 4.

El conjunto de caras que forman el resultado se puede determinar con el siguiente algoritmo:

1. Se dividen las caras iniciales por las aristas de intersección (figura 4).

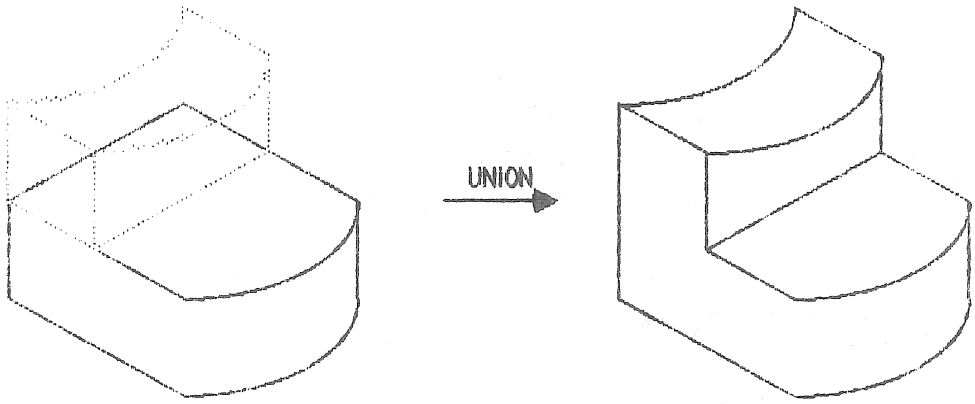
2. El nuevo grupo generado se clasifica para cada sólido en caras dentro, fuera y sobre con respecto al otro sólido, y

3. Se escogen las caras relevantes para formar el resultado dependiendo de la operación que se realiza.

Sean A y B los sólidos iniciales y A_dentro_de_B, A_fuera_de_B, A_sobre_B, B_dentro_de_A, B_fuera_de_A y B_sobre_A las caras clasificadas (A_sobre_B es siempre igual a B_sobre_A). Ninguno de estos conjuntos de caras es cerrado en el sentido de representar un volumen, pero la unión adecuada de algunos de ellos si lo es. En general se cumple que [Manty 1983]:

Operador	Procedimiento para obtener el conjunto resultado
Unión	Pegar (A_fuera_de_B, B_fuera_de_A, A_sobre_B)
Intersección	Pegar (A_fuera_de_B, B_dentro_de_A, A_sobre_B)
Diferencia	Pegar (A_dentro_de_B, B_dentro_de_A, A_sobre_B)

Las operaciones de conjunto generan como resultado un conjunto de sólidos, cada uno bien formado. El resultado de la operación unión es siempre un sólido.



El modelo presentado en esta figura no se puede definir con la técnica de barrido, pero si mediante la operación unión sobre los dos modelos anteriores.

Figura 5.

Requerimientos computacionales.

El modelador desarrollado se puede usar en micro-computadores compatibles con el IBM-PC. Requiere 640 kilo-bytes de memoria RAM, una tarjeta gráfica (ej. Hércules, Ega, etc.) y drivers HALO o GSX. La compatibilidad del software con varias tarjetas gráficas se logró al definir una librería gráfica (GrAndes) que sigue los lineamientos del estandard GKS.

4. CONCLUSIONES

Las decisiones de diseño en el desarrollo de un modelador han sido expuestas, presentando el modelador MGIT-ANDES. Se han combinado varios elementos académicos, para obtener un sistema altamente portable, con el poder de operación suficiente para integrarse a un sistema DAC.

Si bien los operadores de Euler como operadores de construcción del tipo abstaracto de datos, pueden definir por si solos cualquier sólido, la interfase con el usuario es crucial para el modelador interactivo, por tanto es necesaria la investigación continuada en este aspecto.

5. REFERENCIAS BIBLIOGRAFICAS

[Baum 1972]: Baumgart B,G.
Winged edge polyhedron representation.
Stanford Artificial Intelligence Report No. CS-320, octubre de 1972.

[Brai 1975]: Braid I.L.
Notes on a Geometric Modeler.
CAD group document No. 101, laboratorio de computación, universidad de Cambridge, Cambridge, Inglaterra, junio de 1979.

[Brav 1985]: Bravo Mauricio.
Un Modelador Geométrico de Sólidos Basado en los Operadores de Euler.
Documento de Tesis de Grado, Universidad de los Andes, Facultad de Ingeniería, Departamento de Sistemas y Computación, Bogotá, julio de 1985.

[Chiy 1985]: Chiyokura H. y F. Kimura.
A method of representing the solid design process.
IEEE Computer Graphics and Applications, volumen 5, No. 4, abril de 1985.

[East 1979]: Eastman Charles y Weiler Kevin.

Geometric Modeling Using the Euler Operators.
Proc. Fiest Ann. Conf. on Computer Graphics in CAD/CAM systems, MIT, abril de 1979.

[Fumi 1984]: Fumihiko Kimura.
Geomap III. Designing Solids With Free-Form Surfaces.
IEEE Computer Graphics and Applications, volumen 4, no. 6, junio de 1984.

[Gutt 1977]: Guttay J.
Abstarct Data Types and Development of Data of Data Structures.
Communications of the ACM, volumen 20, no. 6, junio de 1977.

[Mart 1982]: Martti Mäntylä y Reijo Sulonen.
GWB: A Solid Modeler with Euler Operators.
IEEE Computer Graphics and Applications, volumen 2, No. 7, 1982.

[Mart 1983]: Martti Mäntylä, Markku Tamminen.
Localizad Set Operations For Solid Modeling.
Computer Graphics, volumen 17, número 3, julio de 1983.

[Mart 1984]: Martti Mäntylä.
A Note on the Modeling Space of Euler Operators.
Computer Vision, Graphics, and Image Processing, volumen 26, 1984, pag 45-60.

[Quir 1986]: Quiroga B. Juan C.
MGIT. Modelador Geométrico Interactivo Tridimensional. Diseño con operaciones de conjunto.

Documento de Tesis de Grado, Universidad de los Andes, Facultad de Ingeniería, Departamento de Sistemas y Computación, Bogotá, Diciembre de 1986.

[Requ 1977]: Requicha A.A.G. y Voelcker H.B.
Constructive solid geometry.
Tech. memo. No. 25, production automation project, Univeristy of Rechester, N.Y. 1977.

[Requ 1980]: Requicha A.A.G.
Representations of rigids solids: theory, methods and systems.
ACM comp. surv. 12 (1980), pag 437-464.

[Requ 1982]: Requicha A.A.G. y Voelcker H.B.
Solid modeling: a historical summary and contemporary assessment.
IEEE Computer Graphics And Applications, 2 (1982) 2, pag 109-144.

[Wils 1985]: P. R. Wilson.
Euler Formulas and Geometric Modeling.
IEEE Computer Graphics And Applications, agosto de 1985.

USO DE HEURISTICOS EN COMPILACION DE UN LENGUAJE DE ALTO NIVEL PARA MAQUINADO EXTERNO EN TORNOS DE CONTROL NUMERICO.

OSCAR EDUARDO RUIZ SALGUERO
RODRIGO LOPEZ BELTRAN

GRUPO DE INVESTIGACION DFAC
FACULTAD DE INGENIERIA
UNIVERSIDAD DE LOS ANDES
APARTADO AEREO 4976-BOGOTA,COLOMBIA

RESUMEN: En la programación de máquinas herramientas de comando numérico (CNC) se usa un lenguaje de bajo nivel que es fijado por normas internacionales. La programación de una pieza a ser realizada en este lenguaje es tediosa y propensa al error, por lo cual se han desarrollado lenguajes de alto nivel que liberen al programador de este esfuerzo. Partiendo de una gramática subconjunto de uno de ellos (APT: Automatically Programmed Tools) suficiente para una descripción de geometrías maquinables por torneado externo, se contruyó un compilador , el cual uso en su módulo de generación de código heurísticos que toman en cuenta la geometría de la pieza y de las herramientas involucradas. Este prototipo, construido dentro de un programa a largo plazo del grupo DFAC (Diseño y Fabricación Apoyados por Computador), para desarrollo de software dirigido a apoyar las labores productivas en la industria, contó con la colaboración de los ingenieros José T. Hernández y Alberto García en las etapas de diseño y realización. Este artículo informa sobre las experiencias y especiativas que dicho proyecto representa.

Palabras Clave: Programación CNC, APT.

1. ANTECEDENTES.

Las máquinas herramientas de comando numérico (máquinas CNC) trabajan con una entrada de bajo nivel especificada según normas de la EIA (Electronic Industries Asociation). Tal entrada, como todo lenguaje de bajo nivel implica riesgo de error en la programación y, por el volumen de cálculos implicado, la labor de programación se hace más tediosa y propensa al error. Lo anterior llevó, en la década de los 50's a desarrollar, en MIT, un primer propotipo de APT (Automatically Programmed Tools), un lenguaje de alto nivel para programación de maquinas CNC. Actualmente existe una gran cantidad de tales lenguajes, muchos desarrollados por las propias industrias, que describen movimientos en tres dimensiones, y cuyo procesamiento implica validaciones sintácticas y geométricas en una primera aproximación, una posterior generación de un codigo intermedio no adecuado para máquinas CNC reales, y por último, una adecuación de dicho código por medio de un post-procesamiento, a una máquina herramienta especifica.

Por la extensión y complejidad del lenguaje y las labores de validación, los ambientes de programación CNC usualmente requieren computadores grandes, lo cual, al lado de la máquina CNC misma, representa altos costos, no accesibles a las empresas del medio colombiano. Una simplificación del problema, con un subconjunto de APT, ha sido abordado en la Universidad de Los Andes, buscando la creación de una herramienta de programación CNC barata y eficaz teniendo en cuenta las necesidades de la industria nacional. Tal herramienta está concebida como ayuda geométrica a la labor de programación CNC en torneado externo. Para simular el uso de diferentes tornos se usa el programa SITOR[1] igualmente construido en la Univesidad de Los Andes, sobre el cual se ejecuta el código producido por el compilador

Este compilador presenta algunas particularidades debidas a las caracteristicas impuestas por el ambiente de programación CNC. Como compilador, podemos decir que sus rutinas de generacion de código no siguen el esquema clásico estático, en el cual, dado un programa fuente y un compilador, hay una sola salida posible como código intermedio o como codigo final[2]. En dichos esquemas clásicos, la generación de un código intermedio es automática, de acuerdo a las instrucciones del programa fuente. Por ejemplo, una instrucción de asignación, luego de validaciones sintácticas y (en algunos casos) de tipos es traducida en un código intermedio en el

---

[1] El Simulador Gráfico de Tornos de Control Numérico: Una Herramienta de Aprendizaje. Memo de Investigación Nº 16, Diciembre de 1986, Centro de Investigaciones de la Facultad de Ingeniería (CIFI).

[2] Aún si se considera la labor de optimización de código, ésta es ejecutada en una forma estática, y funciona casi como un traductor.

cual se expresa la asignación como una transferencia de contenidos entre direcciones en la memoria del computador, y, una posterior pasada, traduce ese código intermedio hacia una forma ejecutable en una máquina particular.

A diferencia de ello, en este caso el módulo generador del compilador debe utilizar heurísticos para decidir el código a generar, estando dichos heurísticos directamente relacionados con las características geométricas de la pieza a maquinar, y con las herramientas disponibles para ello.

Siendo una herramienta que permite simular las variables geométricas del proceso de torneado, y cuyo código producido se ejecuta sobre un simulador de máquinas virtuales, es posible reducir a dos las tres etapas anteriormente descritas involucradas en programación CNC. Se divide, por lo tanto, el proceso de compilación en dos módulos. El primero de validación sintáctica y geometrica, cuyo salida es un conjunto de órdenes de maquinado de alto nivel, que representan la descomposición del problema original en sectores más elementales. El segundo de generacion, aborda en un orden heurísticamente "bueno" la serie de órdenes de maquinado anteriormente producidas y, dentro de cada una de estas tareas elementales emplea otros heurísticos para atacar el problema de generación de una trayectoria que produzca la cavidad deseada dada una o mas herramientas posibles.

En este documento se presentará una visión general de la labor de compilación, comenzando por la gramática base y sus implicaciones sobre la capacidad semántica de las entradas al compilador (sección 2) y continuando con la labor de generación de código haciendo enfasis en las estrategias y heurísticos usados para tal labor, explicados a través de un ejemplo (seccion 3). Por último se concluirá sobre la experiencia obtenida, (sección 4), dando algunas alternativas interesantes que pueden ser exploradas en el futuro.

2. GRAMATICA.

Como gramática básica del compilador, se escogió inicialmente un subconjunto restringido de APT, suficiente únicamente para describir el contorno de la pieza deseada. No se incluyeron instrucciones de loops, subrutinas, etc. Posteriormente se aumentó el conjunto de instrucciones para introducir algunos parámetros de maquinado relevantes para la labor de generacion de código.

DEFINICION.

A continuación se define la gramática en la cual se basa el compilador. Esta gramática, aun cuando es prácticamente un subconjunto de APT, tiene modificaciones que la diferencian de el , y cuya justificación se encontrará mas adelante.

```
‹programa_apt› »» partno ‹ident›
                  machin/ ‹ident›
                                ‹bloque_inic›
                                ‹geometria›
                                ‹param-maquinado›
                                ‹movimiento›
                                stop
                                fini

‹bloque_inic› »» lenght / ‹numero› ,
                 diam / ‹numero›

‹geometria› »» {‹ident› = ‹entidad›}

‹entidad› »» ‹punto› | ‹linea› | ‹circulo› | ‹ident› | ( ‹entidad› )

‹punto› »» point / ‹descr_punto›

‹descr-punto› »»  ‹numero› , ‹numero› |
                  int of , ‹entidad› , ‹entidad› |
                  center , ‹entidad›

‹linea› »» line / ‹entidad› , ‹resto_linea›

‹resto_linea› »» ‹entidad› |
                 (left | right) , tanto , ‹entidad› |
                 atangl , ‹numero› , ‹entidad›

‹circulo› »» circle / ‹descr_circulo›

‹descr_circulo› »» ‹entidad› , ‹entidad› , ‹entidad› |
                   center , ‹entidad› , ‹resto_circulo›

‹resto_circulo› »» ‹entidad› |
                   radius , ‹numero›
```

‹param_maquinado› »» **coolnt / on**
[**neutral /** ‹herramienta›]
‹herramienta›
‹veloc_cuchilla›
‹arranque›
‹veloc_husillo›
‹margen›

‹herramienta› »» **tool /** ‹ident›

‹veloc_cuchilla› »» **fedrat /** ‹numero›

‹veloc_husillo› »» **spindl /** ‹numero›

‹margen› »» **margin /** ‹numero›

‹arranque› »» **cut /** ‹numero›

‹movimiento› »» **from /** ‹entidad›
‹instr_inic_mov›
{‹instr_interm_mov›}
‹instr_final_mov›

‹instr_inic_mov› »» **go / to ,**
‹entidad› ‹resto_inic›

‹resto_inic› »» □[1] |
, ‹modif_mov› , ‹entidad›

‹instr_interm_mov› »» ‹herramienta› |
‹veloc_husillo› |
‹veloc_cuchilla› |
‹margen› |
‹desplazamiento›

‹desplazamiento› »» ‹direccion› /
‹entidad› ,
‹modif_mov› ,
‹entidad›

‹direccion› »» **gorgt** |
goback |
gofwd |
golft

‹modif_mov› »» **to** |
past |
tanto

‹instr_final_mov› »» **goto /** ‹entidad›

---

[1] □ - nulo

Debido a que la anterior gramática realiza una descripción de la geometría de la pieza a través de una declaración de entidades geométricas (‹entidad›) y de un recorrido de la frontera, pero no arroja ninguna luz sobre el proceso de maquinado, el compilador deberá hacer inferencias sobre la estrategia de maquinado a aplicar.

La gramática anterior garantiza que para cada elemento de la frontera de la pieza existen los siguientes atributos de maquinado:

- Velocidad de rotación del husillo en terminado (**spindl**).
- Velocidad de avance de la herramienta en terminado (**fedrat**).
- Herramienta de terminado asociadada (**tool**).
- Margen de desbastado antes de terminación (**margin**)

Los anteriores elementos estan asociados con una frontera, pero existen otros atributos inherentes a toda la pieza;

- Arranque por pasada (**cut**).
- Herramienta neutra por defecto especificada por el programador si este considera que labores de desbastado no se podrán realizar con la herramienta asociada a las fronteras (**neutral**). Esta especificación es opcional.

SEMANTICA.

Esta sección trata las validaciones semánticas que se hacen sobre el archivo de entrada y sobre la frontera y sus atributos.

Antes de comenzar vale la pena definir algunos términos para su entendimiento en adelante

-**entidad**: es una figura geométrica; círculo, línea o punto.

-**frontera**: es una porción de una entidad, a la cual además se asocian atributos como velocidad de rotación del husillo, velocidad de terminado, herramienta de terminacion margen entre el desbastado y la terminación y nombre. Al ser una porción de una entidad debe tener información que identifique tanto la entidad de la cual forma parte como la porción de esa entidad que ella abarca.

A continuación se enumeran algunas de las validaciones hechas sobre la frontera. Validaciones sobre la definición de las entidades no se mencionan, aún cuando desde luego, se hacen. Algunos ejemplos de dichas validaciones son: no definir un punto como la intersección de dos rectas paralelas, no definir un círculo con tres puntos colineales, no definir una recta con dos puntos

idénticos,etc.

-Continuidad de la Frontera: Dado que el programa constituye un "viaje" de entidad en entidad, no puede hacerse un paso de una entidad a otra si no se tocan.

- Vectores de recorrido no-positivos en el eje X: En cada punto de cada entidad de la frontera, existe un vector tangente que muestra exactamente la direccion del movimiento que ha descrito a esa porción de la frontera. Si un móvil dibujara el contorno de la pieza, tal como lo describe el programa, en cada momento él tendría un vector de velocidad, tangente en cada punto no singular a la frontera sobre la que viaja.Se pide que dicho vector nunca tenga una componente positiva en la dirección X con lo cual se pueden garantizar condiciones de maquinado externo.

- El primero y último punto de la frontera no pueden estar en el interior del rectángulo que representa el bloque inicial. La explicación es que si se permitiera esto, un trayecto desconocido entre el borde del bloque inicial y tales puntos quedaría indefinido, sin entidad asociada lo cual equivaldría a decir que la herramienta"apareció" allí sin maquinar nada.

- La descripción del movimiento comienza con un origen, expresado por una entidad un punto, al cual se debe retornar luego de la descripción de la frontera.

- En caso de ambigüedad de puntos destino, por ejemplo en la intersección de dos circunferencias, decide inicialmente la dirección (**gorgt**, **gofwd**, **goback**,**golft**), en caso de subsistir ambigüedad, decide el modificador de movimiento (**to**,**past**,**tanto**). Una vez el punto destino se ha definido, se valida según la regla de la componente X no positiva.

Como conclusión de esta sección, podemos decir que el resultado de la validación semantica sobre el programa de entrada arroja una sucesión de porciones diferentes de frontera que respetan continuidad, tangencia correcta de vectores y retorno de la herramienta al final del trabajo sobre una pieza al mismo punto de salida para que las condiciones de comienzo de la siguiente pieza sean idénticas.

3. GENERACION.

Este capítulo pretende explicar el esquema general de generación de código, tanto en los heurísticos usados para definir un orden de maquinado como en aquellos empleados para las tareas elementales de generación arrojadas por el orden escogido.

Se explicará la estrategia de generación de código en base a un ejemplo. Antes de comenzar con el vale la pena definir el término "sector", que será usado intensivamente a lo largo de este capítulo.

> -**sector** es un área limitada por una sola frontera[1] en su lado izquierdo, y por una sola frontera en su lado derecho. Los niveles superior e inferior de dichas fronteras son iguales respectivamente. La base de un sector puede ser nula, en el caso de que los puntos inferiores de sus fronteras se toquen, o ser una recta horizontal en caso contrario. El tope de un sector siempre es una recta horizontal. Un sector libre es aquel en el cual la frontera derecha coincide con la recta límite derecha del bloque inicial.

3.1 EJEMPLO.

Con este ejemplo se ilustrará la transformación de la información que posee el archivo de entrada hacia una serie de fronteras correctamente definidas desde el punto de vista de maquinado externo, para posteriormente desembocar en una estructura de datos, cuya misma topología conlleva información sobre el proceso de generación.

3.1.1 CODIGO DE ENTRADA.

En seguida se mostrará un programa típico realizado con el conjunto de instrucciones definidas.

```
(zona de instrucciones al postprocesador)
        partno p1
        machin / emco
(zona de declaraciones geométricas)
        length/20 , diam/20
        stpoint = point/21.14
        p1 = point/16.6
        p2 = point/ 12,6
```

---

[1] Para la definición de frontera ver sección de SEMANTICA.

```
p3 = point/16,10
p4 = point/12,2
p8 = point/4,6
p7 = point/8,10
l5 = line/p1,p2
l1 = line/p2,atangl,45,l5
l2 = line/p3,atangl,90,l1
p9 = point/int of, l2,l5
c1 = circle/p2,p3,p9
l0 = line/p7,left,tanto,c1
l4 = line/p4,left,tanto,c1
c3 = circle/p2,p7,p8
p6 = point/center,c3
l6 = line/p7,atangl,90,l0
p5 = point/int of, l1,l6
l11 = line/p8,rigth,tanto,c3
l7 = line/p5, (point/4,10)
l9 = line/(point/int of, l7, l5),atangl,90,l5
l3 = line/p4,left,tanto,c3
l10 = line/(point/2,0),atangl,0,l11
c2 = circle/(point/int of,l7,l5),(point/8,8),(point/10,6)
l8 = line/p5,left,tanto,c2
c4 = circle/center,(point/12,8),radius,2
c6 = circle/center,p2,radius,4
c5 = circle/center,(point/5,6),radius,1
l15= line/(point/0,8),left,tanto,c2
(zona de especificación del maquinado)
coolnt/on
neutral/tool/svvbn3225p16
tool/pdjnr3232m15q
fedrat/90
cut/1
spindl/908
margin/0
(zona de movimientos)
from/ stpoint
go/to,p9
```

```
margin/0.06
fedrat/50
tool/pdjn13232m15q
spindl/890
fedrat/190
gorgt/c1,to,l5
spindl/1000
margin/0.4
golft/l15,to,l2
gorgt/l2,to,c1
tool/pdjnr3232m15q
golft/l1,to,p2
tool/pdjn13232m15q
gorgt/(line/p2,right,tanto,c2),tanto,c2
gofwd/c2,tanto,l5
gofwd/l15,to,l7
gorgt/l7,to,l0
(zona de finalización)
goto/stpoint
stop
fini
```

Antes de comenzar con la discusión de la generación se deben notar varias cosas usando el programa mostrado:

1- La zona de instrucciones al postprocesador da las especificaciones de la maquina herramienta sobre la cual se desea ejecutar el programa.

2- Se garantiza, antes de la instrucción from/stpoint , por la existencia de la zona de especificación del maquinado, que hayan sido precisados los parámetros de fabricación que dirigirán la labor de generación de código.

3- La zona de movimientos tiene por objeto describir univocamente el contorno de la pieza. Es claro que, dada una serie de entidades geométricas, innumerables piezas pueden ser constituidas por ellas; por lo tanto es la zona de movimientos la encargada de quitar la ambigüedad.

4- Entre dos especificaciones de movimiento, en la zona de movimientos, los parametros de maquinado pueden ser cambiados cuantas veces se desee, quedando vigente la ultima especificación.

5- El parámetro cut no puede ser establecido más que una vez; la razón es que él no está asociado a

ninguna frontera sino al desbastado global de la pieza. Igual consideración vale para la herramienta neutra por defecto (**neutral/tool..**), la cual es designada para la totalidad de la pieza.

6- La definición geométrica dada es exagerada para la pieza finalmente hecha; el propósito era únicamente mostrar las capacidades de definición y anidamiento de definiciones, tanto en la zona de declaración geométrica, como en la zona de movimientos o ejecucion.

7- La definición de herramienta neutra por defecto (**neutral/tool**.....) no obliga a que alguna frontera tenga asociada dicha herramienta, ni tampoco lo impide.

8- La zona de finalización marca el final del contorno de la pieza, el retorno al punto inicial y la entrada a un estado latente de la máquina, en espera de la siguiente ejecucion del programa

3.1.2 INFORMACION EXTRAIDA DEL CODIGO FUENTE.

El código fuente provee una descripción de entidades geométricas (figura 3.1) y una forma de recorrido sobre ellas, la cual engendra una serie de fronteras con sus correspondientes atributos (figura 3.2), una lista de herramientas propuestas para el maquinado, y unos parametros generales como el arranque por pasada (**cut/**) y la herramienta neutra por defecto (**neutral/tool**).

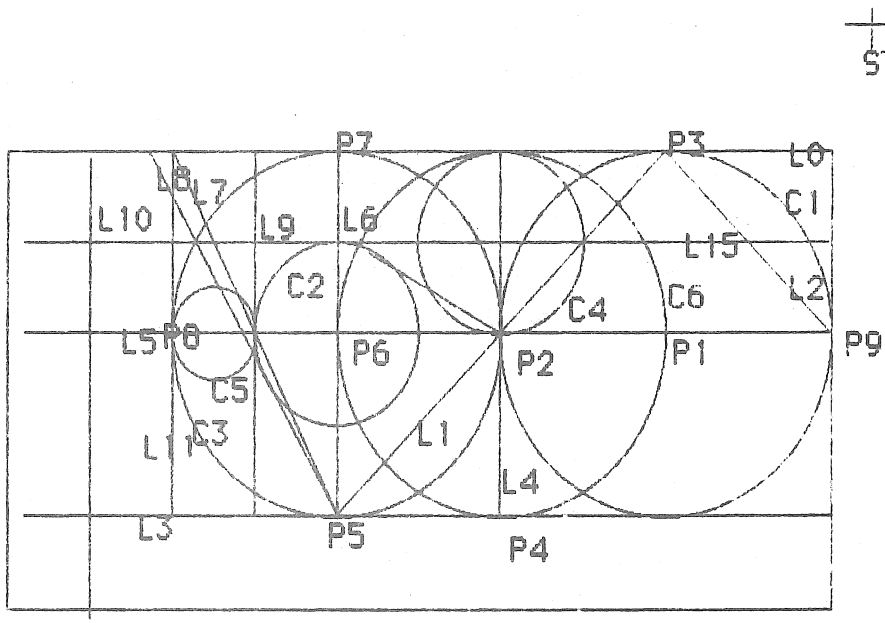


fig 3.1

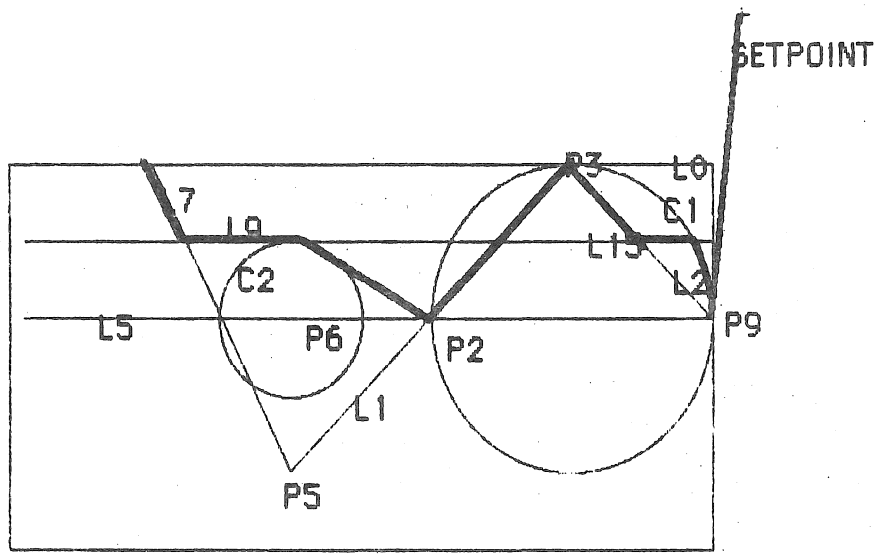


fig 3.2

3.1.3 PARTICION HORIZONTAL DEL TRABAJO DE GENERACION.

Conceptualmente, el trabajo de partición horizontal tiene como entrada una sucesión de fronteras, las cuales forman un contorno válido en cuanto a continuidad y en cuanto a los supuestos de maquinado externo explicados en la sección de semántica del programa fuente. Su salida es una serie de franjas horizontales, cuyos límites son coordenadas Y en las cuales se ha producido un cambio de frontera. Cada franja secciona a todas y cada una de las fronteras con las que se involucra, engendrando nuevas fronteras, más pequeñas. Las fronteras involucradas con una franja son aquellas cuyos límites en Y contienen o son iguales a los límites de la franja (note que una frontera nunca tendrá sus límites estrictamente dentro de una franja por la forma misma de concebir las franjas).

En al figura 3.3 se pueden ver las siguientes tres franjas:

-Limites de la franja.Y_superior = coordenada Y de P3

Y_inferior = coordenada Y de intersección de L15 con L2.

Fronteras involucradas: L2, L1 (seccionada), L7.

-Limites de la franja: Y_superior = coordenada Y de intersección de L15 con L2.

Y_inferior = coordenada Y de intersección de C2 con "?".

Fronteras involucradas: C1 (seccionada), L1(seccionada), C2.

-Limites de la franja: Y_superior = coordenada Y de intersección de C2 con "?".

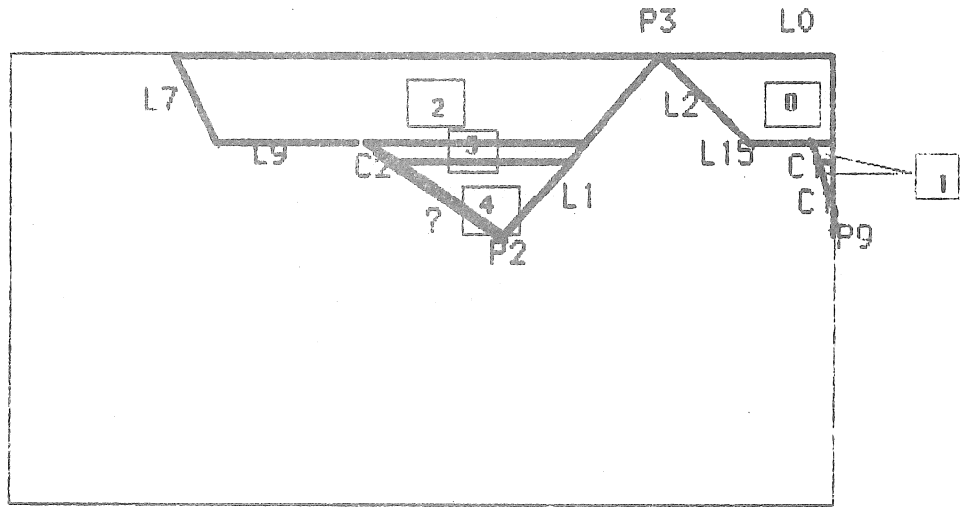


fig 3.3

Y_inferior = coordenada Y de P2.

Fronteras involucradas: C1(seccionada), L1(seccionada), "?".

El propósito del anterior seccionamiento es tener lugares geométricos definidos por **una** frontera al lado izquierdo y por **una** frontera al lado derecho, es decir sectores (ver definicion). Por ello se engendraron las fronteras más pequeñas necesarias para que todos los lugares geométricos así formados tuvieran propiedades de sector.

3.1.4 CONSTRUCCION DEL ARBOL DE MAQUINADO

El árbol de maquinado se construye en base a la definición de **hijo de un sector**;

> **hijo de un sector** es un sector, situado en la franja inmediatamente inferior, y cuyas coordenadas X están contenidas en el rango de coordenadas X del padre

Lo anterior implica que un sector puede tener varios hijos, que serán hermanos entre sí, y se organizarán según coordenadas X descendentes.

El árbol de maquinado es un árbol n-ario, aún cuando en este caso su grado máximo sea 2. El árbol construído con este ejemplo se muestra en la figura 3.3. La numeración dada muestra el orden de recorrido del árbol. El nodo raíz no se identifica con ningun sector Las relaciones entre los sectores dibujados son:

- **0** y **2** son hijos de la raíz principal y hermanos entre sí.

- 1 es el único hijo de 0.
- 3 es el único hijo de 2.
- 4 es el único hijo de 3.

Como ilustración a las ideas sobre sectores dadas arriba, notamos que 0 y 1 son sectores libres. 1 y 4 tienen base nula (es un punto), mientras que 0,2 y 3 no. Todos los sectores tienen una horizontal como tope. Un sector siempre tiene una frontera asociada con su lado izquierdo y una frontera asociada con su lado derecho. La creación de la malla por cada punto singular garantiza esto, aún cuando atomize exageradamente el árbol.

Note que, al haber un punto de quiebre de la frontera anónima ("?") hacia C2, este nivel divide innecesariamente el sector1. La labor de normalización del árbol de maquinado incluye reconocer que esos dos sectores en que se ha dividido el sector 1 se pueden fundir en uno solo, puesto que cumplen la definición de sector dada arriba.

El recorrido en pre-orden garantiza que no se intentará maquinar primero el sector 1 que el 0, o el 3 que el 2. La prioridad derecha provoca que 0 sea maquinado antes que 2, aún cuando físicamente ello no es obligatorio. Lo anterior hace que los sectores maquinables por cilindrado tengan prioridad.

3.1.5 ESTRATEGIAS DE GENERACION.

Habiendo establecido la forma en la cual la frontera se transforma en una definición de sectores, organizados en forma de árbol para definir el orden de maquinado,resta explicar cuáles con las estrategias propuestas para cada sector.

3.1.5.1 ESTRATEGIA 1.

Esta se usa para sectores libres; incluye ciclos de cilindrado cuyo arranque es establecido por el parámetro **cut/** del programa, hasta que se llegue a la profundidad suficiente para que solo reste el terminado final. Durante todo el proceso, la herramienta usada es la asociada a la frontera izquierda del sector, puesto que la frontera derecha no existe para el programador; es únicamente un concepto dentro del compilador. Las figuras 3.4 y 3.5 muestran ejemplos de esta estrategia usada para maquinar los sectores 0 y 1.

Las comprobaciones realizadas para determinar si el maquinado de un sector de este tipo es posible son:

- Tangencia correcta en los puntos superior e inferior de la frontera izquierda.
- Corte izquierdo de la herramienta suficiente para realizar el arranque especificado por **cut**.

3.1.5.2 ESTRATEGIA 2.

La estrategia 2 se usa cuando las herramientas de ambos lados son correctas para terminar su respectiva frontera, pero no son capaces de realizar el desbastado, ni terminar la frontera opuesta. En este caso se usa la herramienta neutra por defecto (**neutral/tool**...) para realizar el desbastado, y cuando se ha llegado la fondo del sector, se reemplaza la herramienta neutra por la correspondiente a cada lado, para, alternativamente, terminar la frontera izquierda, y la derecha. La figura 3.6 muestra el esquema de generación usado.

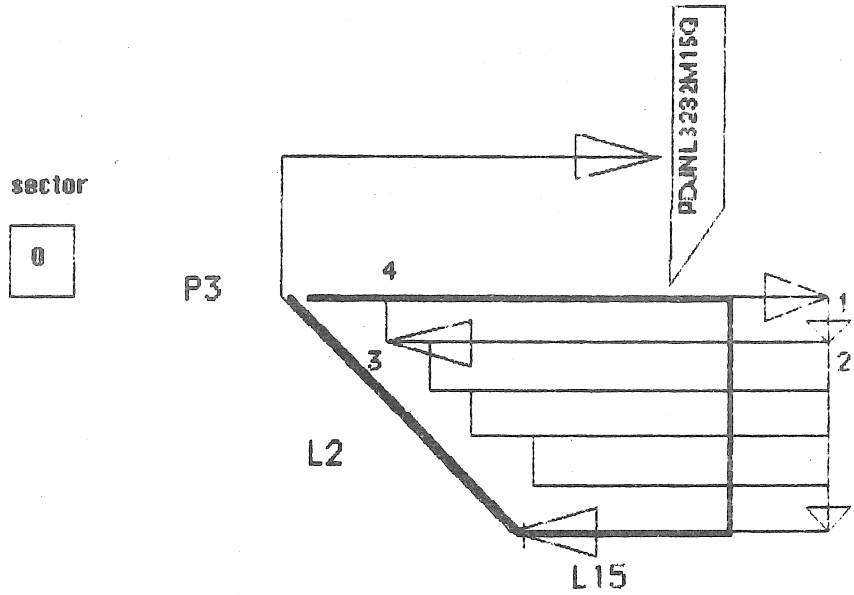


fig 3.4

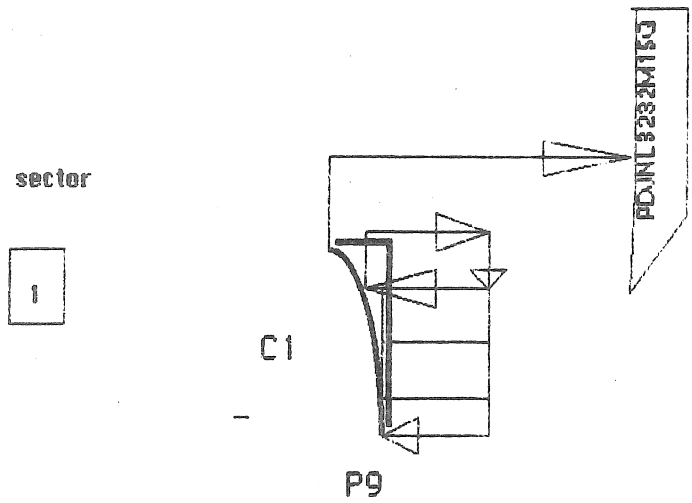


fig 3.5

Cuando se ha hecho el desbastado , un sector parecido al original pero mas estrecho aparece, por efecto de la especificación de márgenes a lado y lado.

Las condiciones necesarias para que este maquinado resulte exitoso son:

- Tangencia y posicionabilidad correctas tanto en el sector final como en el sector intermedio más estrecho que queda por efecto del desbastado.

- La herramienta de desbastado tiene que poder llegar hasta el fondo mismo del sector. La razon es que las dos herramientas de terminación harán su labor de abajo hacia arriba . Por lo tanto deben poder llegar hasta el fondo. La herramienta de desbastado sigue el camino mostrado determinado por los margenes (**margin**) y el arranque por pasada (**cut**). Cuando los puntos de

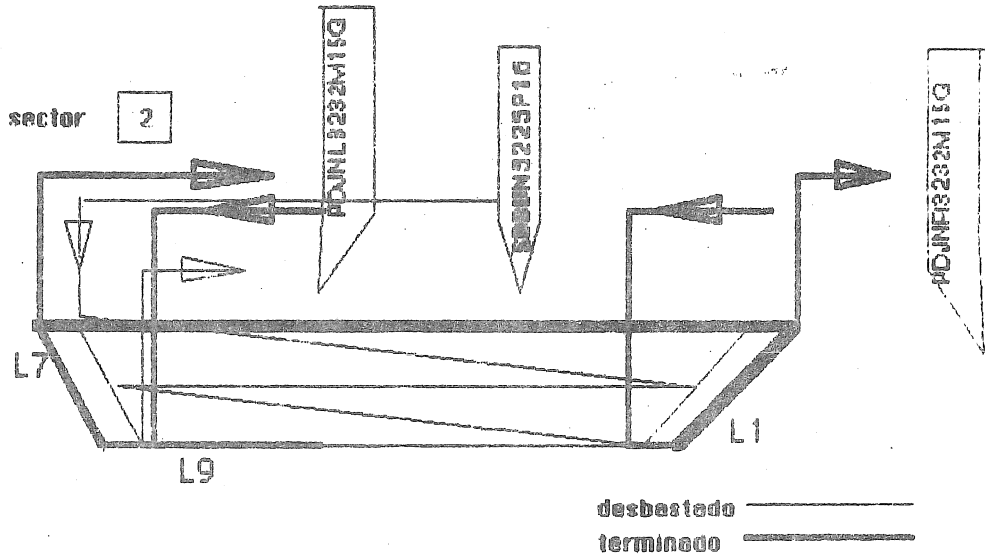


fig 3.6

zig-zag del desbastado están demasiado cerca, la relación distancia-horizontal/arranque se hace muy pendiente, lo cual viola el angulo permisible de maquinado para la herramienta. En este caso se desiste de esta estrategia.

- La herramienta de desbaste debe tener suficiente corte para garantizar el arranque por pasada (**cut**) especificado.

3.1.5.3 ESTRATEGIA 3.

Esta estrategia busca realizar el desbaste y el terminado final de un sector encajonado, con la misma herramienta. Ello implica que la herramienta escogida debe poder terminar tanto el lado izquierdo como el derecho del sector, y, tener un corte suficiente de acuerdo a lo especificado por

CUT. Un ejemplo de ella se muestra en las figuras 3.7 y 3.8.Esta estrategia ensaya tanto con la herramienta de la frontera derecha como con la herramienta de la frontera izquierda, y con el movimiento de "caracol", tanto en el sentido del reloj como al contrario.

Las comprobaciones que se tienen que hacer dentro de este esquema son las siguientes:

- La herramienta escogida debe ser posicionable y de tangente correcta en los puntos superior e inferior tanto de la frontera izquierda como de la derecha. Posicionable quiere decir que colocada allí no choca ninguna frontera. De tangencia correcta, quiere decir que sus ángulos tanto del lado del filo como del contrario permiten llegar a dichos puntos realizando la operación de contorneado necesaria sin dañar la frontera. Aún cuando los dos conceptos parezcan el mismo no lo son; generalmente hay problemas en los límites superiores de un sector por motivo de la tangencia pero no de la posicionabilidad; la herramienta se puede colocar en tal sitio sin danar ninguna frontera; sin embargo su movimiento a lo largo de la frontera la daña.

-El ciclo de desbastado puede terminar antes de llegar al fondo del sector. En tal caso, para que se tenga exito debe restar una profundidad menor al corte especificado por **cut**.

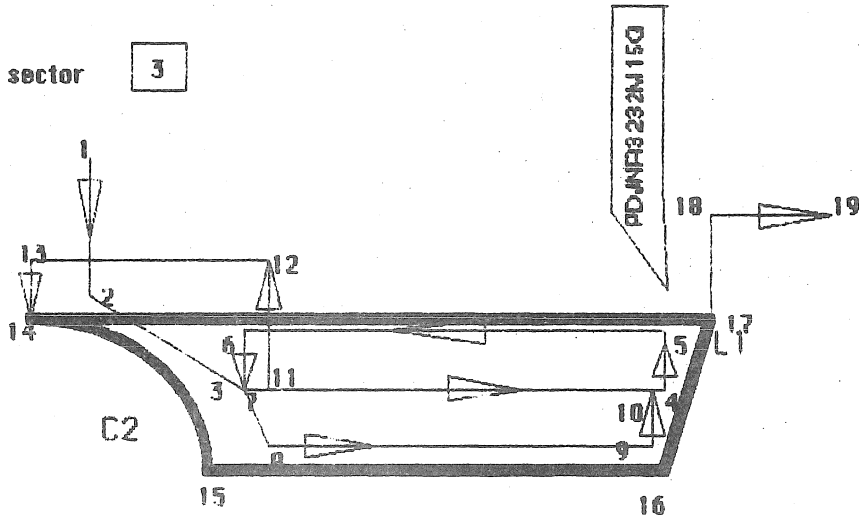


fig 3.7

-Aún cuando la tangencia de la herramienta y su posicionabilidad sean correctas,la trayectoria de un punto a otro puede cortar la frontera. En este caso, lo que ocurre generalmente es que el parámetro de margen (**margin**) es demasiado pequeño. Aumentarlo puede solucionar el problema. Un ejemplo en el cual esto podría pasar es el movimiento 2-3 de la figura 3.7. Allí si el margen de la frontera izquierda fuera demasiado pequeño, el movimiento podría cortar la frontera.

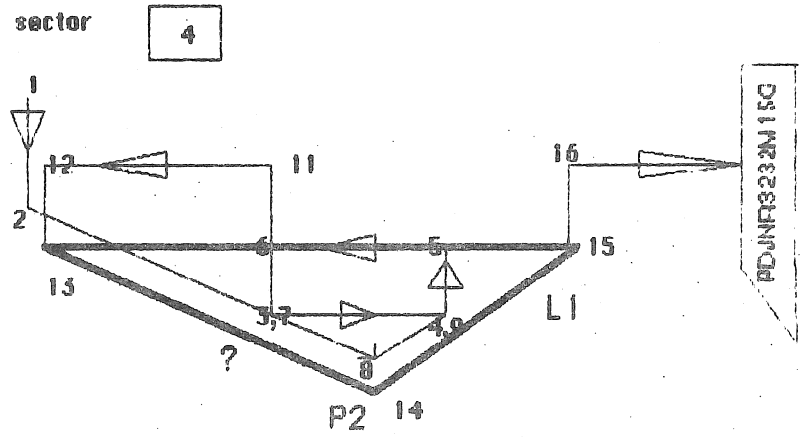


fig 3.8

Para finalizar la sección de generación se hace una precisión sobre la especificacion de la herramienta en el programa. En ambientes de programación CNC normales se puede dar una especificación genérica de la herramienta en compilación, pues su validacion final puede y debe ser hecha en el postproceso, cuando se trata la máquina herramienta particular. Aqui, por el contrario, se espera la especificación de la herramienta desde el comienzo. Lamentablemente, los códigos normalizados para especificación de herramientas no describen unívocamente la geometria de una herramienta, por lo cual, una representación de ella se debe basar en los datos de algún fabricante. El programa generador permite hacer uso de catálogos definidos por el programador, con el fin de no comprometerse con ningún fabricante. El catálogo es un archivo de texto que es procesado para su validación y rápido acceso, y su objeto es precisar totalmente los parámetros geométricos relevantes de la herramienta. Su proceso y contenido, sin embargo no serán tratados en este artículo.

4. CONCLUSIONES.

Este capítulo trata dos aspectos: el primero de ellos establece la comparación entre los objetivos impuestos y las realizaciones finales del proyecto; el segundo de ellos esboza conclusiones acerca de la realización misma y traduce esas conclusiones en sugerencias para ulteriores proyectos.

El propósito inicial de definir un subconjunto de APT para maquinado externo se vio afectado por dos circunstancias: la primera de ellas fué inherente al hecho de que en este proyecto se debía condensar la labor de compilación y postprocesamiento; la segunda fué debida al interés de usar un subconjunto los más pequeño posible, con el fin de explorar hasta dónde se podía llegar en la tendencia de cargar al compilador con la responsabilidad de definir estrategias tanto generales como específicas de maquinado.

El primer punto obedece a lo siguiente: en ambientes normales de programación CNC al disponerse de un postprocesador se deja en el programa de entrada la posibilidad de trabajar con una herramienta genérica, definida únicamente por algunos parámetros relevantes en ese momento, para hacer una validación de cortes y ángulos más exhaustiva en la labor de postproceso. En el caso presente, se requiere que la definición de la herramienta se haga completa desde la entrada inicial. Tal definición se realizó de acuerdo a los standards de maquinado externo, pero ello de todas formas implicó pedir al programador especificar el nombre de la herramienta propuesta, lo cual no ocurre en un preproceso normal APT. Sin embargo esta exigencia no carga al programador con ningún dato que no debiera saber ya en el momento de planear una pieza.

El segundo punto, de reducción en el conjunto de instrucciones básicas, llevó a transladar parte del trabajo que debiera enfrentar el programador hacia el compilador y el generador de código. La posibilidad de ello reside en el hecho de que con el conjunto definido, lo único que el programador hace es describir la frontera asignándole a sus componentes atributos tales como velocidad de maquinado, velocidad de avance, etc. Esto representa un "salto" muy grande entre el programa de entrada y el código generado, que debe ser asumido por el compilador, ya que para una misma descripción de la frontera, existen innumerables programas en código de máquina que la ejecutan. La conveniencia de ello será discutida más adelante.

Esta segunda desviación se manifiesta en tres modificaciones hechas al lenguaje: introducción de un parámetro de arranque por pasada, **cut**; un parámetro de terminación **margin**; y la declaración opcional del programador de una herramienta **neutral** para el caso de requerirse

para la ejecución de la pieza de una herramienta de desbastado neutra además de aquellas de terminación de la frontera en cuestión.

Como se puede ver, las adiciones no comprometen la filosofía de descargar de trabajo al programador que persigue el compilador, puesto que son factores muy comunes de maquinado que él como persona con algún conocimiento en el tema debe saber.

El hecho de que el compilador asuma totalmente la tarea de generación de código sin preguntar nada al usuario tiene consecuencias sobre la consecución de objetivos iniciales del proyecto en lo que tiene que ver con maquinabilidad de la pieza. En este momento, el compilador genera código a través de tres estrategias distintas y cuando no puede hacerlo con ellas declara que la generacion fué imposible. Estas tres estrategias cubren la mayor parte de las piezas torneadas externamente, pero desde luego, pueden existir piezas para las cuales el compilador se declara impotente, sin que ello quiera decir que la pieza , con otra estrategia más sofisticada no se pueda realizar. Sin embargo, por las pruebas hechas se llegó a la conclusion de que tales piezas son verdaderamente infrecuentes.

Habiendo tratado los logros y limitaciones del proyecto, conviene dedicar algun espacio a las conclusiones que trabajar en él ha producido.

En primer lugar vale la pena tratar las consecuencias que un subconjunto muy reducido de instrucciones tiene sobre la facilidad en la generación de código. Como se dijo anteriormente, esta deficiencia produce de inmediato un salto muy grande entre el programa, y lo que finalmente serán las estrategias usadas. El problema no es una cuestión de aumentar el repertorio de generación para que el compilador no "se dé por vencido" pronto, aunque tal acción es deseable. El problema estriba en que no hay una parte del lenguaje que exprese de alguna forma lo que el programador quiere. Tal y como están las cosas ahora el define la frontera las herramientas asociadas, los parámetros de maquinado, y en ese momento pierde el control del proceso. Por otro lado, el compilador realiza todo el trabajo de identificación de primitivas trabajables, ordenamiento sobre ellas y escogencia de la estrategia a seguir. Juzgar correcta la ultima aproximación no es del todo razonable, y obligar al programador a especificar el movimiento de las herramientas lo carga de un trabajo del que , se ha visto en las condiciones de maquinado externo, podría estar liberado. Si lo que interesa es proveer una herramienta que haga el trabajo con una direccion del programador se puede llegar a compromisos como proponer una lista de estrategias para cada sector ordenadas segun razonamiento del programa para que el programador escoja, rechace, o modifique el orden propuesto, o , posiblemente presentar ordenamientos sobre los sectores a maquinar, y luego , dentro de ellos ordenamientos

sobre las estretegias a usar, o , que el programa pida ayuda al experto que tiene al frente cuando sus conocimientos fallaron del todo para un sector, o cuando nota que las estrategias que faltan por probar están muy atrás en la lista de prioridades, lo cual parece indicar poca eficiencia en el caso en particular, etc.

El proceso de generación se puede hacer más "razonable" en el sentido de que conserve su eficiencia en la transformación entrada/salida, pero se sirva de un sistema flexible de heuristicos para la escogencia de la linea de acción a seguir; dicho de otro modo, es posible abordar la labor de generación de código no de una forma estática, sino con un comportamiento adaptable por parte del programa, asimilable a un sistema experto.

5 BIBLIOGRAFIA

[AHO77] Aho Alfred, Ullman Jeffrey, "Principles of Compiler Desing".
Addison-Wesley 1977.

[KOR83] Korem, Yoram, "Computer Control of Manufacturing Systems".McGraw-Hill Book,1983.

[KRA86] Kral Irving, "Numerical Control Programming APT".
Prentice-Hall, 1986

[ROD86] Rodriguez German. "SITOR. Manual del Usuario".
Uniandes 1986.

FLUX2D y FLUX3D SOFTWARES DE CAD PARA LA MODELIZACION DE CAMPOS ELECTROMAGNETICOS

**Orietta SANTANA, Gérard MEUNIER,
Jean Louis COULOMB, Jean Claude SABONNADIERE**

**Laboratoire d'Electrotechnique de Grenoble (UA CNRS 355)
ENSIEG, BP.45
38402 St. Martin d'Hères
FRANCIA**

RESUMEN

*La utilización de métodos numéricos para modelar (**modelling**) fenómenos electromagnéticos, nos ha permitido la realización de los softwares FLUX2D y FLUX3D en el Laboratoire d'Electrotechnique de Grenoble.*

FLUX2D (2 dimensiones y simetría de revolución) y FLUX3D (3 dimensiones) son softwares para el cálculo de campos electromagnéticos que utilisan el método de los Elementos Finitos para resolver ecuaciones de derivadas parciales. Estos softwares comprenden 3 partes :

***El Pre-procesador** que recibe las informaciones que describen la geometría y las propiedades físicas del modelo en forma interactiva. Los elementos de la malla son generadas automáticamente (elementos triangulares en 2D o tetraédricos en 3D) a partir de los nodos que se apoyan en la geometría (fronteras del objeto).*

***El Procesador** de resolución, que calcula los valores de las incógnitas en cada nodo de la malla o mallaje (**mesh**). Los elementos finitos son rectilíneos (primer orden) o curvilíneos (segundo orden). Las no-linealidades a causa de materiales no- saturables son tratadas con el método de Newton-Raphson. El sistema de ecuaciones lineales es resuelto con el método ICCG (Incomplet Conjugate Cholesky Gradient).*

***El Post-procesador** que es concebido para analizar, evaluar y presentar gráficamente los resultados del procesador de cálculo. Así, podemos visualizar líneas isovalores (2D) o superficies isovalores (3D) para magnitudes escalares; una red de flechas para magnitudes vectoriales y todo sobre planos de corte (cross section) o sobre las fronteras del objeto. También es posible de evaluar magni- tudes globales como fuerzas y momentos que se ejercen en una región o en todo el objeto.*

I. INTRODUCCION

Las primeras aplicaciones de CAD (Diseño Asistido por Computador) vieron la luz en el campo de la electrónica donde la necesidad de concebir circuitos impresos y luego integrados a muy grande escala ha hecho indispensable el uso de la informática. Su empleo ha sido rápidamente repandido en la aviónica (CAD de la Industria de la Aviación) donde la complejidad de la concepción de aviones o helicópteros necesita desde el punto de vista mecánico y de estructuras el empleo de softwares bastante elaborados. Poco a poco las técnicas de CAD han ganado campos diversos como la industria del automóvil, la arquitectura, la concepción de cableajes electrónicos. Hoy en día, nuevas aplicaciones se desarrollan en todos los campos de la industria.

La industria eléctrica no se ha escapado a este fenómeno. El decenio próximo verá sin lugar a dudas la generalización de tales sistemas de los cuales las primeras implantaciones han demostrado por parte de los usuarios un gran interés.

En la concepción de estructuras electromagnéticas, el conocimiento del campo magnético es necesario para predecir:

- los fenómenos locales (densidad de flujo magnético, densidad de corriente, densidad de fuerza, ...) que caracterizan el funcionamiento interno de las máquinas;
- los parámetros globales (reluctancias, momentos, ...) que conciernen el comportamiento de las máquinas en una cadena de conversión electromecánica.

Antiguamente, la distribución del campo en las máquinas eléctricas era establecida con métodos analíticos lineales, transformaciones conformes y métodos de circuitos magnéticos lineales. Los resultados obtenidos con estos métodos eran mejorados utilizando factores correctores empíricos que tomaban en cuenta las no-linealidades. Sin embargo, la colmplejidad geométrica creciente así como la carrera hacia la optimización, nos probaron la necesidad de métodos de análisis más rigurosos.

Los computadores han favorecido el desarrollo de métodos numéricos. El método de las diferencias finitas fué el primero en ser empleado. Bajo este método, las ecuaciones de derivadas parciales representando el problema del campo son remplazadas por un conjunto de ecuaciones en diferencias finitas expresadas en los nodos de la red de puntos repartidos en el objeto modelado. Este procedimiento genera un conjunto de ecuaciones lineales, que una vez resueltas, dan una distribución aproximada del campo. Este método ha sido bastante utilizado a pesar de la dificultad de implantación en un software CAD a causa de los problemas de discretización inherentes al método.

El método de los elementos finitos se basa en una formulación variacional de las ecuaciones de derivadas parciales representando el problema del campo. Ini-

cialmente utilizado en análisis de estructuras mecánicas y en transferencia térmica, permite la resolución de problemas bidimensionales de electrostática , de magnetostática y de magnetodinámica, lineales o no, en sistemas cartesianos o de simetría de revolución. Este método ha sido implantado con éxito en los softwares de CAD, donde FLUX2D desarollado en el *Laboratoire d'Electrotechnique de Grenoble*, es una ilustración, gracias a la relativa facilidad de discretización de todo el dominio bidimensional en elementos finitos y gracias a la capacidad de resolver los problemas no-lineales.

La extrapolación del método de los elementos finitos al caso tridimensional, encierra ciertos problemas como:
- la multiplicidad de formulaciones posibles
- la dificultad para discretizar objetos tridimensionales
- la dificultad para visualizar los resultados obtenidos.

Sin embargo, el método de los elementos finitos es el único método que actualmente permite resolver problemas tridimensionales no lineales en una óptica general de CAD. FLUX3D es una prueba de esta realización.

II. PRE-PROCESADOR

El papel que desempeña este este módulo es la adquisición de datos y la preparación de las informaciones para el tratamiento del problema con el método de los elementos finitos. El pre-procesador de los softwares FLUX2D y FLUX3D realiza las siguientes funciones:
- parametrización
- descripción geométrica del objeto
- mallaje
- preparación del problema.

A continuación presentaremos las funciones del pre-procesador de FLUX3D ya que el de FLUX2D es más sencillo y está comprendido en el primero.

1. Parametrización

La experiencia adquirida en la utilización de softwares elementos finitos nos ha mostrado la importancia de la descripción parametrizada de la geometría. Generalmente un usuario de un programa corno FLUX3D no estará satisfecho de su primer cálculo y deseará modificar una parte de su modelo geométrico (el espesor del entrehierro de un contactor, por ejemplo) para luego poder calcular un nuevo campo magnético y analizar los nuevos resultados físicos.
La parametrización debe permitir al usuario poder visualizar en forma sencilla con respecto a una característica importante de las especificaciones, el efecto de la variación de uno o varios parámetros constructivos del objeto. Por ejemplo, dada una pieza eléctrica y una magnitud fija que se desea obtener, se debe poder jugar

con las cantidades de material (fierro, cobre, ...) o las medidas de la pieza. En general se trata de optimizar el volumen y la cantidad de material.

2. Descripción geométrica del objeto

Un programa interactivo nos permite crear objetos a partir de la característica más elemental: las coordenadas de los **puntos** que constituyen las fronteras del objeto (figura 1). A esta fase la llamamos *geometría*. Enseguida viene la fase de *topología* que consiste en crear también de forma interactiva las **líneas** que pueden ser :

- segmentos de recta (entre 2 puntos)
- arcos de círculo (entre 3 puntos).

Luego son creadas las **facetas** del objeto a partir de una lista de líneas consecutivas, estas facetas pueden ser:

- planas, poligonales (a partir de segmentos, mínimo 3)
- planas, no poligonales (a partir de segmentos y/o arcos)
- cilíndricas (mínimo 2 segmentos y 2 arcos).

Los **volúmenes** son finalmente creados con la unión de varias facetas.

Las estructuras de datos empleadas en el caso de FLUX3D para esta etapa son las siguientes :

```
tipo_entidad PARAMETRO :
   NUMERO          entero [1] ;
   NOMBRE          caracteres [16] ;
   VALOR           real [1] ;
fin_entidad ;

tipo_entidad PUNTO :
   NUMERO          entero [1] ;
   COORDENADAS     real [3] ;
   FORMULA         caracteres [80] ;
fin_entidad ;

tipo_entidad LINEA :
   NUMERO          entero[1] ;
   selección entre:
      SEGMENTO     PUNTO [2] ;
      ARCO         PUNTO [3] ;
   fin_elección ;
fin_entidad ;
tipo_entidad FACETA:
   NUMERO          entero [1] ;
   LINEAS          LINEA [2:99] ;
   ORIENTACION     entero [2:99] ;
   COLOR           caracteres [16] ;
fin_entidad ;
```

```
tipo_entidad VOLUMEN:
   NUMERO          entero [1] ;
   FACETTAS        FACETTA [4:99] ;
fin_entidad ;
```

donde *tipo_entidad* corresponde a la estructura **record** del Pascal. Cada *tipo_entidad* se identifica con un nombre y una serie de atributos que pueden ser enteros, reales y/o asociaciones hacia otras entidades (**link**). Por ejemplo, LINEA.SEGMENTO es una asociación hacia 2 entidades PUNTO. Más detalles en cuanto a la estructuración de datos del pre-procesador se encuentran en [2].

3. Mallaje

La generación de la malla consiste en determinar una cantidad de nodos y de elementos finitos que formen una discretización aceptable del objeto. Esta discretización se hace generalmente en 5 etapas:

1) Discretización automática de las líneas.
2) Triangulación de cada faceta del objeto (algoritmo de Delaunay 2D).
3) Discretización de las facetas del objeto (algoritmo de Delaunay).
4) Tetraedrisación (algoritmo de Delaunay 3D).
5) Creación de nodos internos al volumen (algoritmo de Delaunay).
6) Regularización automática de nodos (algoritmo de Laplace).

La discretisación de líneas se realiza de forma automática. Los nodos son repartidos de forma uniforme (la distancia entre nodos es la misma) o concentrada a una extremidad (exponencial).

El algoritmo de Delaunay genera a partir de un grupo de nodos iniciales repartidos de forma cualquiera en el espacio, un conjunto de elementos triangulares (2D) o tetraédricos (3D). Estos elementos tienen la característica de ser los *más equilaterales posibles* y ademas la esfera circunscrita a un elemento dado (a los 4 nodos de un tetraedro por ejemplo) no contiene ningún otro nodo. Más detalles sobre este algoritmo en [1].

Así en la etapa 2 se aplica el algoritmo de Delaunay a cada faceta del objeto a partir de los nodos creados en las líneas. En la etapa 4 los nodos iniciales son los nodos de las facetas del objeto.

Las etapas 3 y 5 consisten en insertar nuevos nodos correctamente posicionados en los elementos *muy grandes* , es decir en los que el volumen excede a un cierto volumen deseado, o en los elementos que no respetan las especificaciones de calidad (angulos muy agudos, tetraedros planos). Cada nuevo nodo se inserta en el centro de gravedad del elemento, un mallaje local es entonces construido alrededor del nodo.

Para la última etapa se aplica el algoritmo de Laplace [5]. Cada nodo que no forma parte de una frontera es desplazado hacia el centro de gravedad de los nodos que le son conectados. Este procedimiento es simple y converge rápidamente (2 o 3 iteraciones).

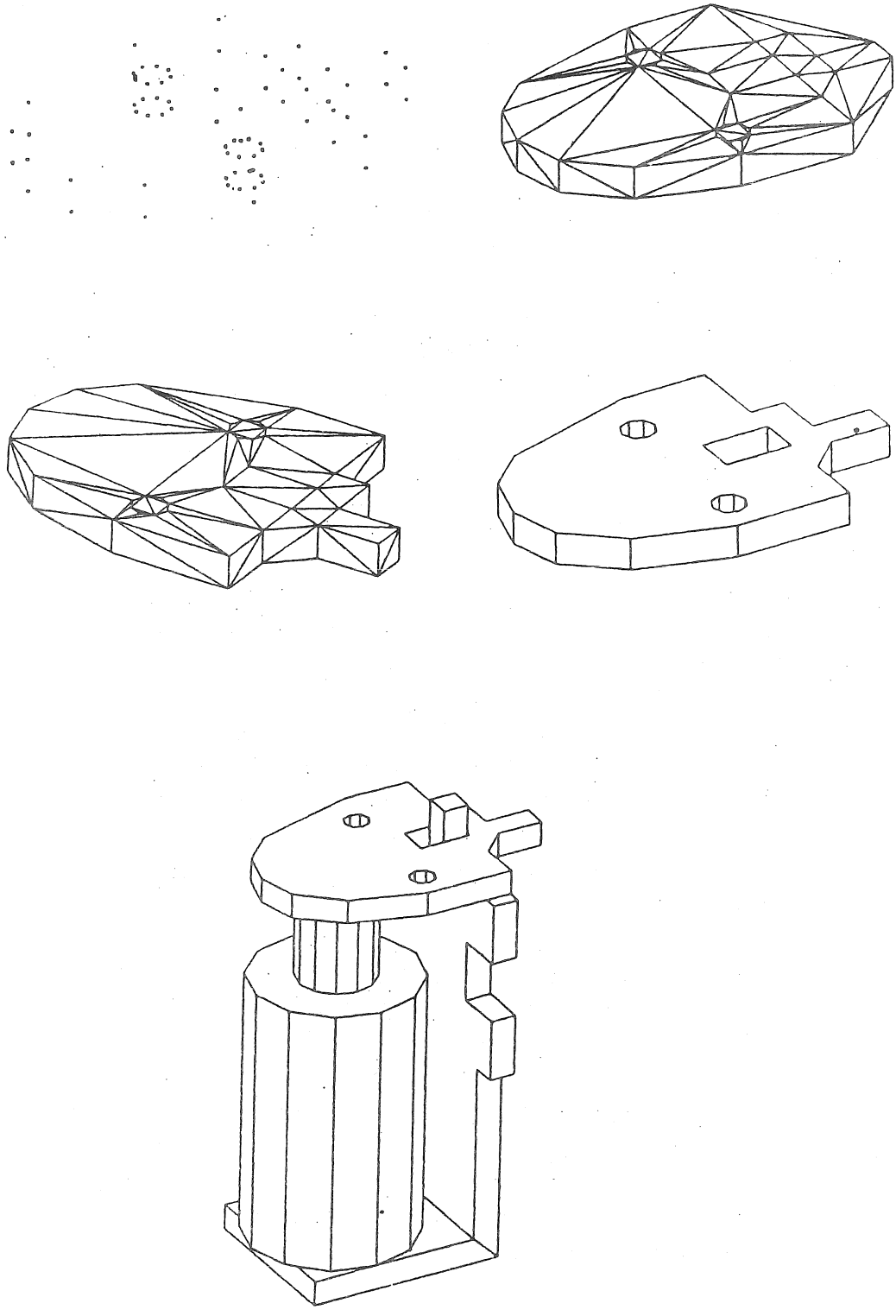
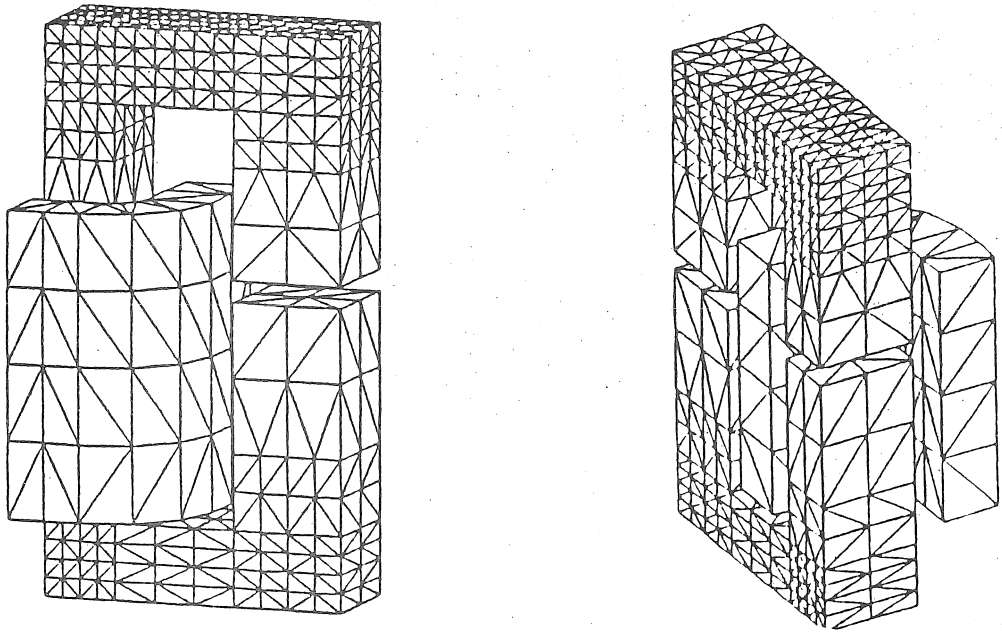
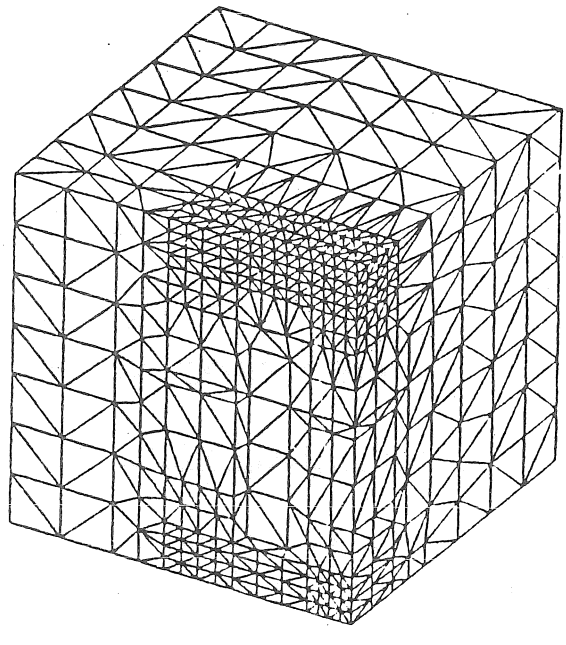


Figura 1.

Proceso de construcción de un objeto.



MAILLAJE EN ELEMENTOS FINITOS
DE UN CUARTO DE CONTACTOR ELECTROMAGNETICO



MAILLAJE DEL CONTACTOR Y DEL AIRE QUE LO ENGLOBA

Figura 2.

4. Preparación del problema

La preparación del problema consiste en la operación de definición de regiones y fronteras:

- descripción de las propiedades del material (por ej. conductividad térmica, permitividad, ..)
- descripción de las fuentes (por ej. fuente de corriente)
- descripción de las condiciones en los límites
- descripción de las condiciones iniciales para los problemas dependientes del tiempo.

III. PROCESADOR

El procesador de cálculo permite resolver problemas de:

- electrostática
- magnetostática escalar
- magnetostática vectorial
- magnetostática mixta
- conductor
- electrocinética

La estructura del procesador es tal que una nueva formulación puede ser fácilmente implantada. Acepta los elementos finitos de primer o segundo orden, rectilíneos o curbilíneos, tetraédricos, prismáticos o hexaédricos. Las no-linealidades dadas por la presencia de materiales saturables son resueltas con el método de Newton-Raphson.

Los sistemas de ecuaciones algebraicas de matrices dispersas symétricas definidas positivas son resueltos con el método de los gradientes conjugados con pre-condicionamiento por descomposición de Cholesky incompleto (ICCG). Los otros sistemas son resueltos con el método de los bigradientes conjugados con pre-condicionamiento.

IV. POST-PROCESADOR

1. Evaluación de una cantidad local utilizando formulas

Un post-procesador que permite la evaluación de cantidades locales a partir de resultados brutos del cálculo con elementos finitos puede ser construido siguiendo:

- una técnica por procedimiento: el programa permite el cálculo de un conjunto de cantidades predefinidas y nada más que ellas. El principal inconveniente de esta técnica es la necesidad de aumentar nuevas instrucciones al interior del programa para implantar la evaluación de una cantidad no prevista inicialmente.

- una técnica por fórmula: el usuario expresa las cantidades deseadas bajo la forma de una expresión aritmética (por ej. F = J x B) en función de cantidades de base tales como las coordenadas, las variables de estado y las características físicas. Este método permite efectuar todos los cálculos deseados sin modificar el programa que puede ser idéntico sea cual fuera la formulación y la dimension. Este método es el que ha sido implantado.

Administración de parámetros

El post-procesador permite el manejo interactivo de parámetros (estos parámetros son diferentes a los que fueron descritos en el pre-procesador). Un parámetro se caracteriza por:

- el número de componentes (1 para un escalar, 3 para un vector)
- su tipo (real o complejo)
- su origen que puede ser:
  - variables de estado (A,Φ,...)
  - características físicas (J,V,...)
  - derivada de variables de estado (gradiente, rot,div)
  - coordenadas de espacio (X,Y,Z)
  - fórmula (combinación algebraica de parámetros exixtentes)
  - cantidad resultante de una integración global.

Biblioteca de sub-programas para la evaluación de fórmulas

Esta biblioteca es una extención en el caso vectorial o complejo de una biblioteca existente utilizada para la manipulación de fórmulas escalares por los pre-procesadores. Ella permite la evaluación de fórmulas construidas a partir de las funciones de base (seno, coseno, raiz cuadrada, ...) y los operadores usuales (+,-,*,/,**). Cada expresión contiene uno o varios parámetros (por ej. SQRT((NU*B-HC)**2) o MOD(NU*B-HC)).

Utilisación de fórmulas en el post-procesador

La base de datos de los parámetros ha sido inicializada dinámicamente y automáticamente con las variables de estado y las características físicas correspondiente a la formulación en curso. Por ejemplo, en el caso de un problema tridimensional de magnetostática vectorial, tenemos los parámetros iniciales siguientes:

A : potencial vector, variable de estado, 3 componentes reales
J : densidad de corriente, característica física, 3 componentes reales
NU : reluctividad magnética, característica física, 1 componenete real (isotrópico)
HC : campo coercitivo, característica física, 3 componentes reales
X,Y,Z : coordenadas de espacio

El usuario puede construir cualquier fórmula utilizando los parámetros precedentes. Esta fórmula podría convertirse en un nuevo parámetro para ser utilizado en nuevas fórmulas. Por ejemplo:

- creación de un nuevo parámetro B = rot (A)
- creación de un nuevo parámetro H = NU*B-HC
- evaluación de MOD(H)

En un punto dado, la evaluación de una fórmula necesita las siguientes etapas:

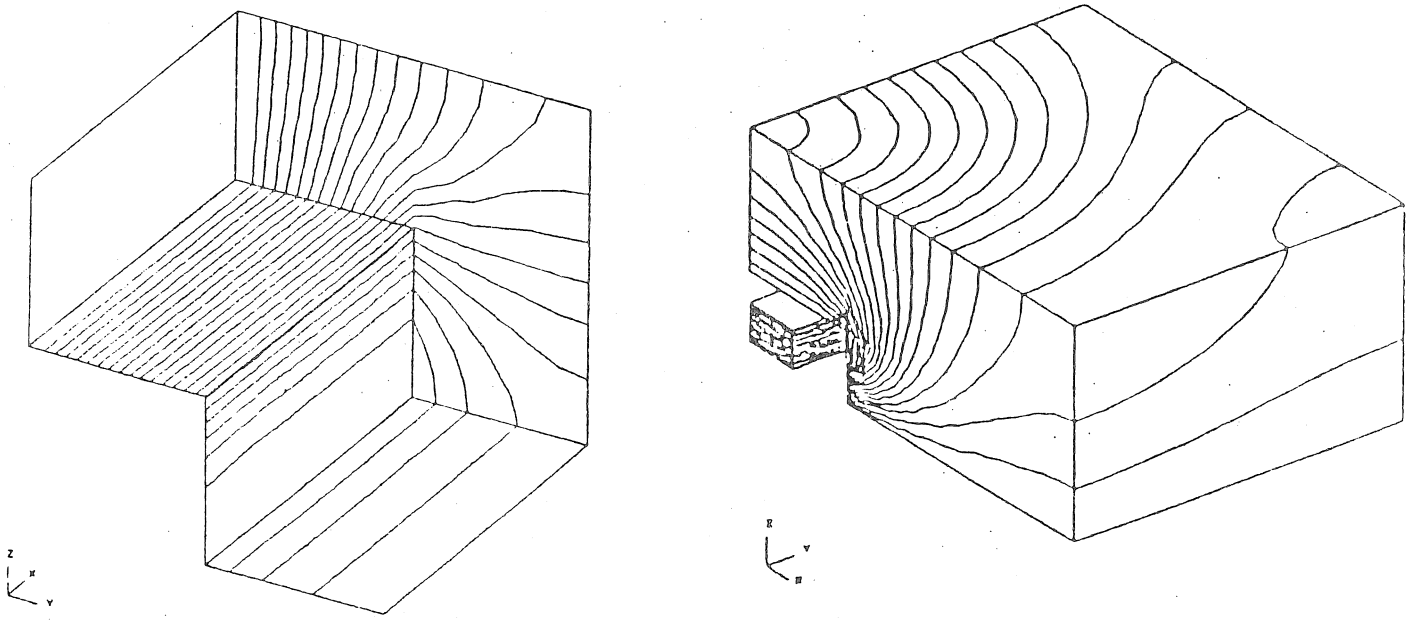
- entrada interactiva y analisis sintáxico y semántico
- determinación de la lista jerárquica de parámetros utilizados (un parámetro puede depender de otro parámetro)
- cálculo de parámetros utilizados en función de su origen
- evaluación gracias a los útiles de la biblioteca.

2. Visualización de variables de espacio utilizando fórmulas

Las variables de espacio (representadas con fórmulas) pueden ser visualizadas en dos tipos diferentes de soporte (definido de forma interactiva): *secciones planas o fronteras de un objeto*. En los 2 casos, la superficie se representa de manera interna por un conjunto de triángulos (intersección del plano con los elementos finitos, o descomposición de fronteras) que nos permiten eliminar las líneas escondidas *(hidden lines)*. El usuario puede pedir:

- una representación de curbas de nivel de una magnitud escalar (figura 3).
- une reperesentación con flechas de una magnitud vectorial (figura 4).

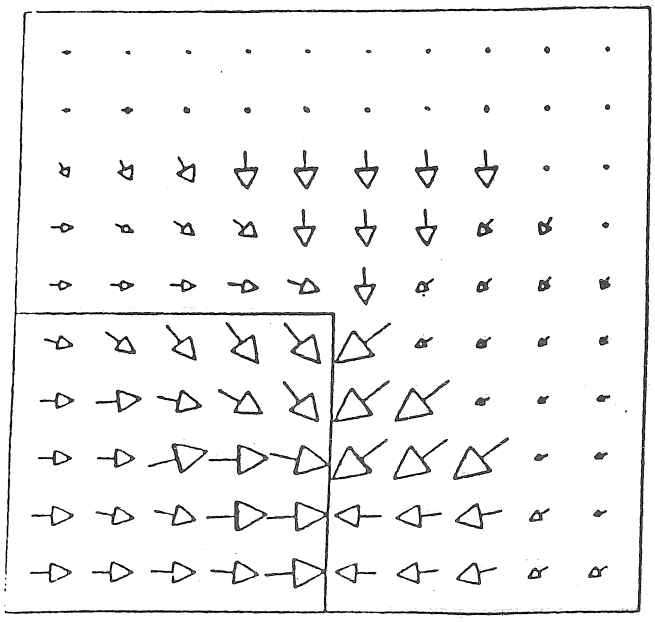
Además el usuario puede pedir la representación bajo la forma de grafo (X,Y) una magnitud a lo largo de una línea arbitraria (figura 5).



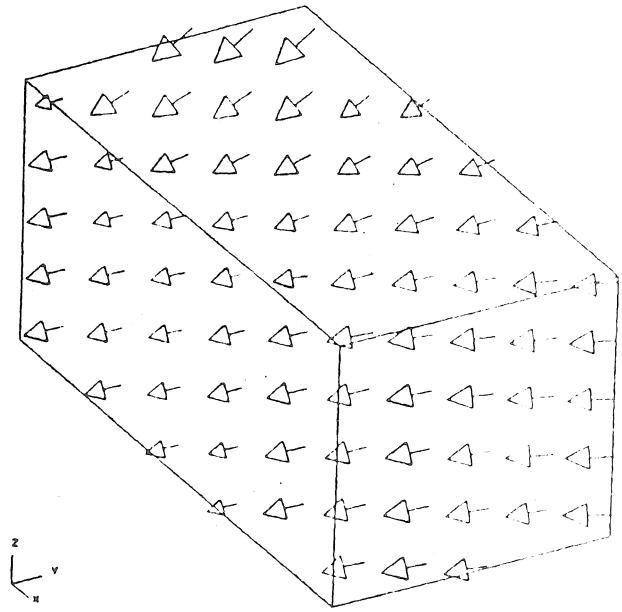
EQUIPOTENCIALES

Figura 3.

FLUX2D

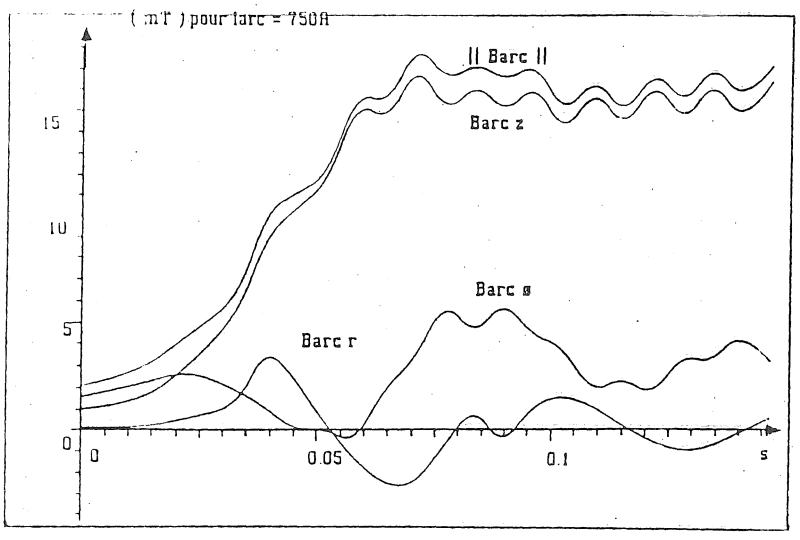


FLUX3D

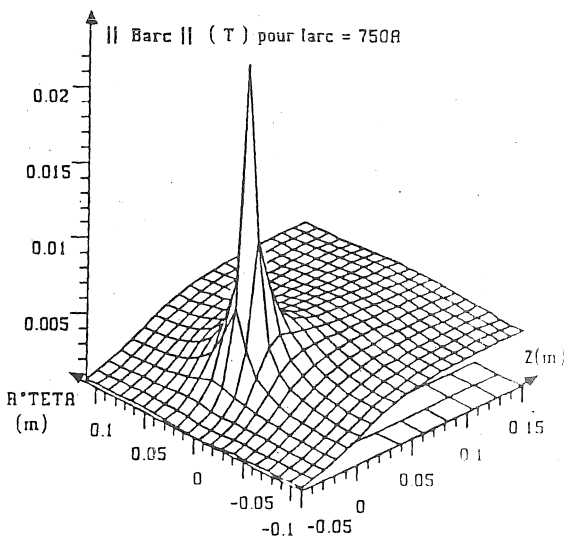


Red de flechas.

Figura 4.

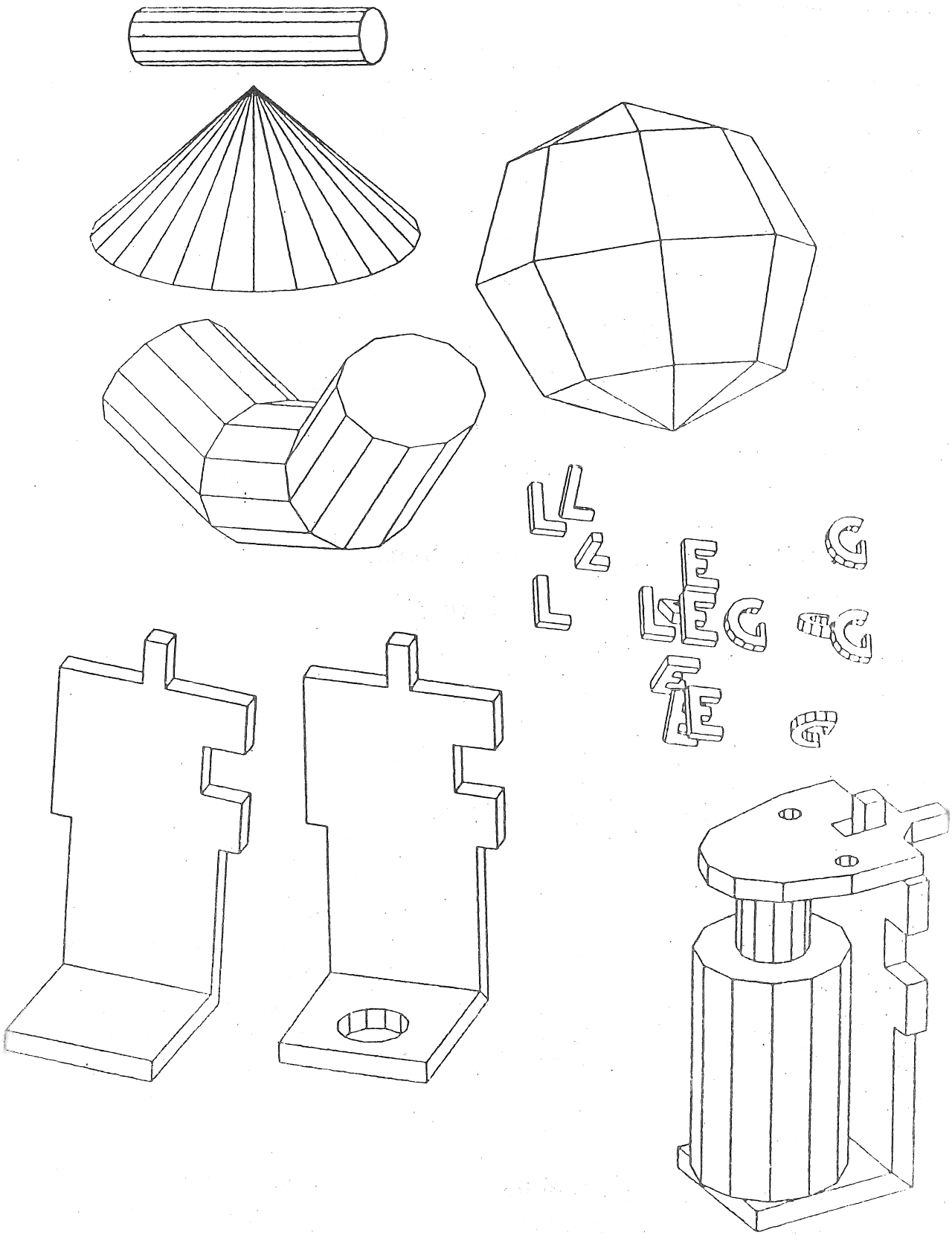


Variación de una magnitud a lo largo de un segmento.



Variación de una magnitud a lo largo de un rectángulo.

Figura 5.



Visualisación de objetos eliminando líneas escondidas (hidden lines)

Figura 6.

3. Cálculo de cantidades globales utilizando las fórmulas

El post-procesador brinda la posibilidad de calcular interactivamente las integrales de una fórmula. La cuadratura concierne el objeto entero o una región en particular. Sobre cada elemento finito se utiliza una integración numérica con el método de Gauss. Por ejemplo:

- la integral de 1 da el volumen;
- la integral de H x B/2 da la energía .

El resultado de una integración puede ser almacenado opcionalmente como parámetro para una utilización futura.

BIBLIOGRAFIA

[1] *Y. Du Terrail, O. Santana, G. Meunier, JL. Coulomb, GET3D: un mailleur en elements finis tetraedriques pour la modelisation des champs electromagnetiques. Artículo presentado a STRUCAD'86, Paris, Octubre 1986.*

[2] *O. Santana, JPh. Iafrate, JL Coulomb, A data structure for a finite element package. Artículo presentado a COMPUMAG'87, Graz-Austria, Agosto 1987.*

[3] *JC. Sabonnadière, J.L. Coulomb, Eléments finis et CAO. Paris, Ediciones Hermes, 1986.*

[4] *G. Meunier, J.L. Coulomb, D.Savalle, Use of formulæ in finite elements post-processing. Artículo presentado a COMPUMAG'87, Graz-Austria, Agosto 1987.*

[5] *JPh. Iafrate, O. Santana, JL. Coulomb, DB-LIB: a tool box of data description and management. Artículo presentado en ICIAM'87, París, Junio 1987.*

[6] *Jean-Louis Coulomb, Analyse tridimensionnelle des champs électriques et magnétiques par la méthode des éléments finis. Tesis de Docteur es-sciences physiques , INP de Grenoble, Junio 1981.*

[7] *Yves Du Terrail, Modélisation géométrique et topologique en 3 dimensions pour l'application de la méthode des éléments finis en electromagnétisme. Tesis de Doctor del INP de Grenoble, Octubre 1985.*

[8] *Didier Savalle, Contribution à la conception et à la réalisation d'un processeur d'exploitation et de visualisation des résultats d'un calcul de champs électromagnétiques tridimensionnels. Tesis de Doctor del INP de Grenoble, Mayo 1987.*

C O M P U T A C I O N G R A F I C A:

METODOS DE APROXIMACION DE CURVAS Y SUPERFICIES

Por: Daniel Formica

Claudio Delrieux

Daniel Masni

Laboratorio de Sistemas Digitales
Depto. Ingeniería Eléctrica
Universidad Nacional del Sur
ARGENTINA

Palabras Clave: Paquetes Gráficos Interactivos; Algoritmos de Aproximación e interpolación de Curvas y Superficies; Modelado por Puntos de Control; Representación Gráfica Tridimensional.

I - INTRODUCCION

En las representaciones de gráficos por computadoras, gran cantidad de casos involucran figuras cuya definición no es necesariamente funcional (no podemos evaluar las posiciones del gráfico en base a variables), sino más bien proviene de un conjunto de datos encontrados experimentalmente (representación de objetos reales, funciones no conocidas pero medibles, etc.). En dichos casos se puede modelar la figura a representar según dos técnicas opuestas: se reconstruye la figura obteniendo una gran base de datos experimentalmente de modo que los programas clásicos graficadores de funciones nos sigan siendo de utilidad, o bien utilizamos algún método en nuestro programa que nos permita reconstruir la figura aproximandola a partir de pocos puntos experimentales.

Este último método, llamado método de los PUNTOS DE CONTROL, es unánimemente utilizado para el modelado de curvas y superficies no funcionales, ya que la otra alternativa, además de requerir bases de datos de tamaños imprácticos, produce en general problemas de actualización y de resultados gráficos que la hacen inelastica y ensorrosa.

Dada la necesidad de implementar una rutina de aproximación de curvas y superficies para un paquete gráfico interactivo tridimensional, se investigaron, ensayaron y compararon cuatro técnicas, basadas todas en el modelado por puntos de control, que fueron los métodos de Hermite, Bezier, B-Spline y Beta-Spline.

Los requerimientos para poder incluir dicha rutina en el paquete gráfico existían la capacidad de producir gráficos tridimensionales interactivos de gran calidad y realismo, con la mayor rapidez y versatilidad posible, sin depender del Hardware utilizado. El criterio de selección estuvo orientado por las siguientes características [2]:

1. INDEPENDENCIA RESPECTO DE LOS EJES: La forma resultante de un objeto no debe cambiar si los puntos de control se miden con distintos sistemas de coordenadas (invariancia a rotación, traslación y cambios de escala)

2. VALORES MULTIPLES: Debe ser permitida la representación de funciones multivaluadas respecto de cualquiera de los ejes.

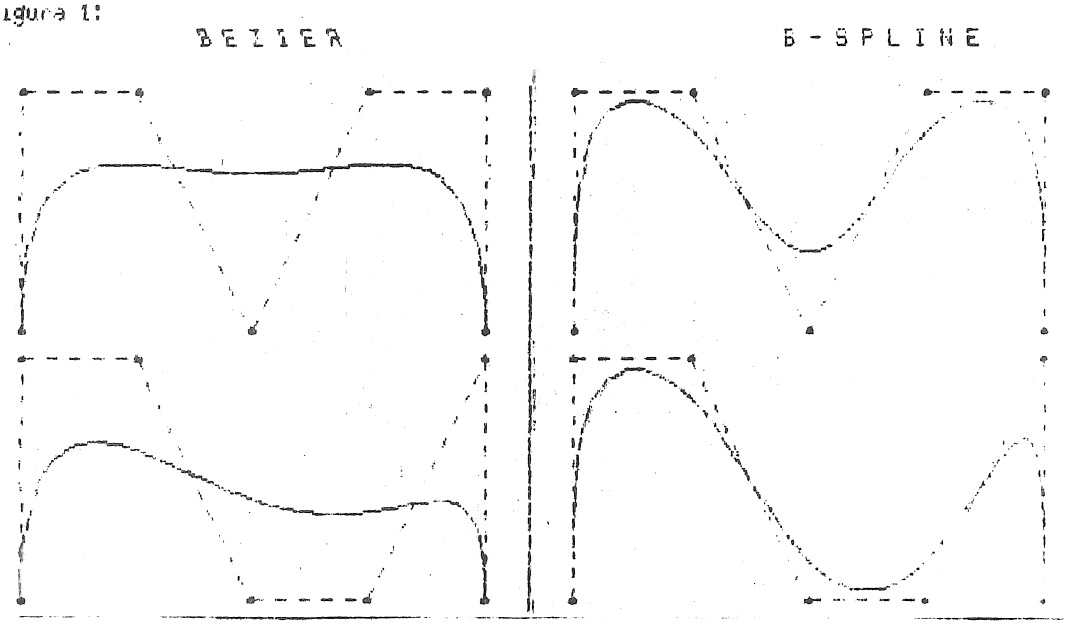
3. PROPIEDAD DE DISMINUCION DE VARIACIONES: La representación debe suavizar y nunca amplificar las pequeñas irregularidades delineadas por los puntos de control.

4. CONTROL LOCAL O CONTROL GLOBAL: Si al alterar un punto de control se modifica solo un entorno de la curva original, denominaremos CONTROL LOCAL a la característica del método. En oposición un algoritmo con CONTROL GLOBAL modificará toda la curva al alterarse un punto de control cualquiera (ver Figura 1). Debe ser conocida la característica de control del método utilizado para poder seleccionarlo respecto de la aplicación.

5. VERSALTILIDAD: La propiedad del método de poder modificar el resultado gráfico a voluntad sin introducir cambios en la base de datos.

6. ORDEN DE CONTINUIDAD: Sea f(x) considerada como la derivada de orden cero de si misma. El orden de continuidad "i" de la curva es aquella f'(x) del grado máximo que sea derivable. El orden de continuidad de la curva provista por el método debe ser conocido y en lo posible podrá modificarse.

Figura 1:



II - HERMITE

El desarrollo de **Hermite** [3] parte de las ecuaciones polinómicas resultantes de suponer continuidad de orden cero y de orden uno. El método consiste en recorrer los puntos de control tomándolos de a pares junto con una estimación de las derivadas de la curva en dichos puntos. Como resultado el algoritmo entrega una curva que pasa por los puntos de control con las derivadas especificadas interpolando las posiciones y derivadas intermedias, cuyos valores quedan determinados por los polinomios mencionados.

Las condiciones establecidas son:

(1) $U(0) = P_{0,u}$
$U(1) = P_{1,u}$

(2) $U'(0) = P_{0,u}$
$U'(1) = P_{1,u}$

donde: U denota cada uno de los ejes coordenados (x,y,z)
U(i) es el polinomio que realiza la interpolación entre las coordenadas $P_{0,u}$ y $P_{1,u}$.

La condición (1) asegura continuidad de orden cero y la condición (2) asegura continuidad de orden uno. Planteando dichas condiciones la ecuación de una curva que interpola un conjunto de N puntos es:

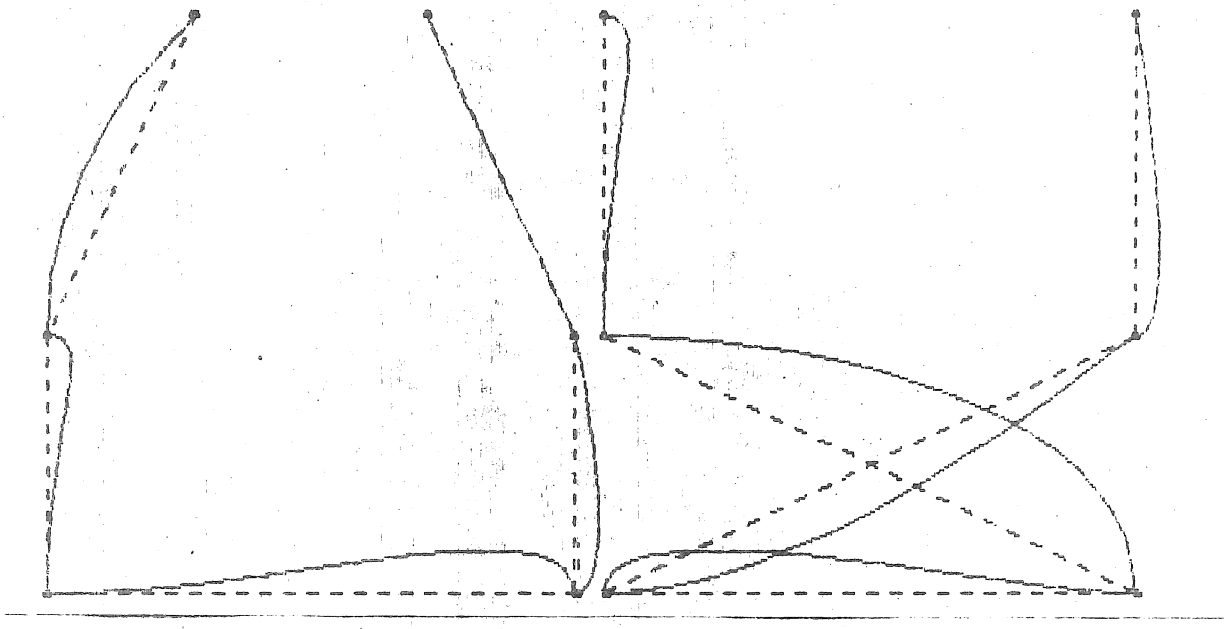
$$P(u) = \sum_{i=1}^{N} \sum_{j=0}^{1} \sum_{k=0}^{1} s_{j\,k}(u) \cdot P^{j}_{i-1+k}$$

donde $s_{JK}(u)$ son los polinomios cúbicos de Hermite:

$$s_{oo}(u) = 2.u^3 - 3.u^2 + 1$$
$$s_{o1}(u) = -2.u^3 + 3.u^2$$
$$s_{1o}(u) = u^3 - 2.u^2 + u$$
$$s_{11}(u) = u^3 - u^2$$

En la figura 2 vemos el comportamiento de este método (graficado en trazo lleno) para dos poligonos de control dados (graficados en líneas punteadas).

Figura 2: Hermite



III - BEZIER

El algoritmo de Bezier [1] construye polinomios de orden de continuidad "n", siendo "n" el número de puntos de contol. Utilizando una función estadistica denominada "Familia de Polinomios de Berstein" [4] obtenidos a partir de la distribución binomial construye las llamadas FUNCIONES DE MEZCLA de modo que la posición ocupada por un punto determinado de la curva es el resultado de "mezclar" la posición de TODOS los puntos de control en proporciones indicadas por las funciones de mezcla. Por ello su caracteristica es de ser control global. La ecuación de una curva representada de este modo es:

$$P(u) = \sum_{i=0}^{n} B_{i,n}(u).P_i$$

donde u es el parámetro tal que:

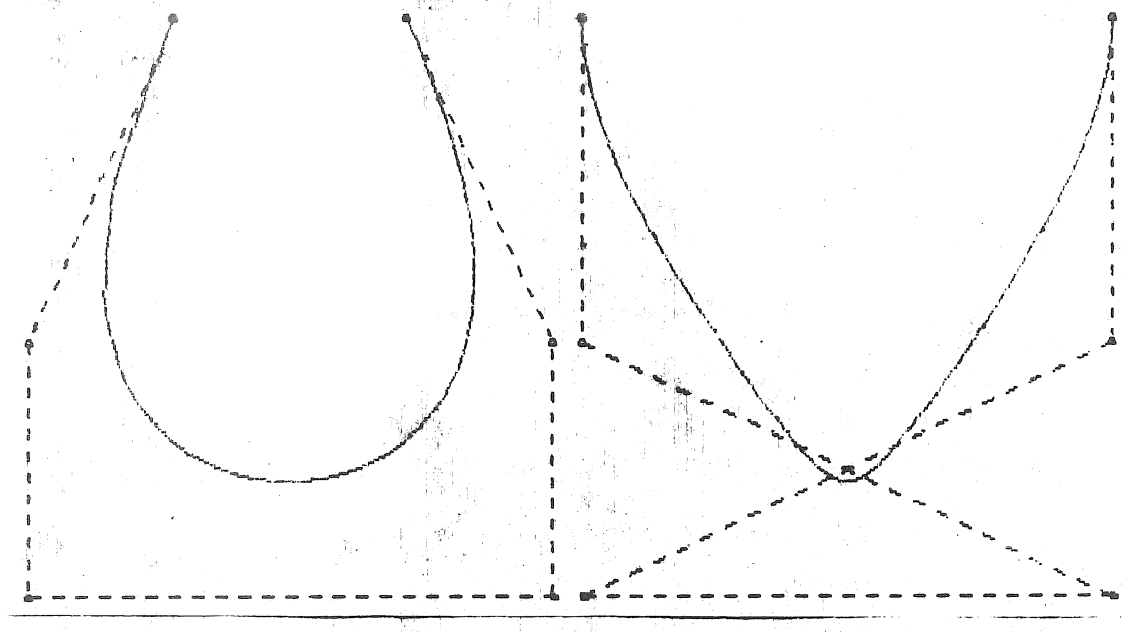
$$0 \leq u \leq 1$$

y la Función de Mezcla es:

$$B_{i,n}(u) = \frac{n!}{i! \cdot (n-i)!} \cdot u^i \cdot (1-u)^{n-i}$$

En la figura 3 se observa el resultado obtenido con los polígonos de control anteriores aproximados por este método.

Figura 3: Bezier



IV - B-SPLINE

El B-Spline [1], [2] fue el próximo método investigado. Las curvas Spline se obtienen con una filosofía similar a las curvas de Hermite, pero con ciertas variaciones en el planteo de las condiciones de borde. Un "Spline" se define como la suma de un conjunto de polinomios que describen la curva entre los puntos 'i-1' e 'i' cumpliendo con las restricciones:

(A) $P_i(u) = P_{i+1}(u)$

$$u_i \leq u \leq u_{i+1}$$

$$i = 0, 1, 2, \ldots, k-1$$

(B) $P_i^j(u_i) = P_{i+1}^j(u_i)$

$$J = 0, 1, 2, \ldots, r-1$$

$$i = 0, 1, 2, \ldots, k-1$$

donde k: Número de puntos de control
J: Orden de continuidad del polinomio
r: Número de restricciones

La condición (A) impone que el polinomio 'i' tiene valor cero excepto en el intervalo entre los puntos de control 'i-1' e 'i'. La condición (B) establece que en la unión entre los polinomios se conserva un orden de continuidad 'J'.

Con dichas restricciones la ecuación de una curva que pasa por N puntos de control es:

$$P(u) = \sum_{i=0}^{n} N_{i,k}(u) \cdot P_i$$

Donde $N_{i,k}(u)$ es una función de mezcla para el orden de continuidad k que se expresa en forma recursiva:

$$N_{i,k}(u) = \frac{(u - P_i) \cdot N_{i,K-1}(u)}{P_{i+K-1} + P_i} + \frac{(P_{i+K} - u) \cdot N_{i+1,K-1}(u)}{P_{i+K} - P_{i+1}}$$

donde $N_{i,1} = 1 \quad$ si $P_i \leq u \leq P_{i+1}$

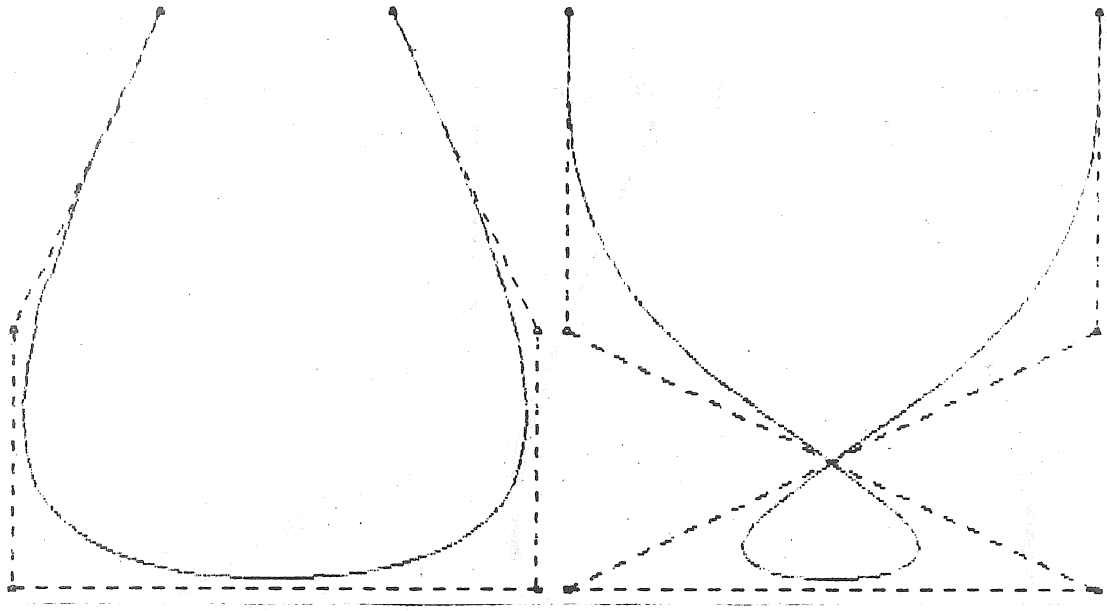
$= 0 \quad$ de otro modo

Estas funciones de mezcla tienen la misma forma, independientemente del valor de i. Por lo tanto, para que la curva pase por el primer y por el último punto de control, se repiten k veces dichos puntos asegurando continuidad de orden cero en los mismos.

Debido a que el ojo humano no detecta fácilmente órdenes de continuidad mayores que dos, se eligió este valor para la implementación del método. Con dicho orden de continuidad solo es necesario evaluar cuatro puntos de control para encontrar la posición de cualquier punto de la curva. Esto indica que, a diferencia del método de Bezier, su característica es de control local (al variar un punto de control cambia solamente un entorno de la curva).

En la figura 4 se observa el resultado obtenido al graficar los polígonos de control anteriores con este ultimo método y k=4 (orden de continuidad dos), siendo observable cómo la curva se aproxima mejor al polígono de control

Figura 4: B - S p l i n e



V - BETA-SPLINE

El Beta-Spline fue el último método investigado [6], [8], [9]. Las curvas Beta-Spline se obtienen, al igual que en el B-Spline, como suma de polinomios, pero planteando las siguientes condiciones de borde [9] en la intersección de cada intervalo:

(D) $$P_{i-1}(u_i) = P_i(u_i)$$

(E) $$B_1 \cdot P'_{i-1}(u_i) = P'_i(u_i)$$

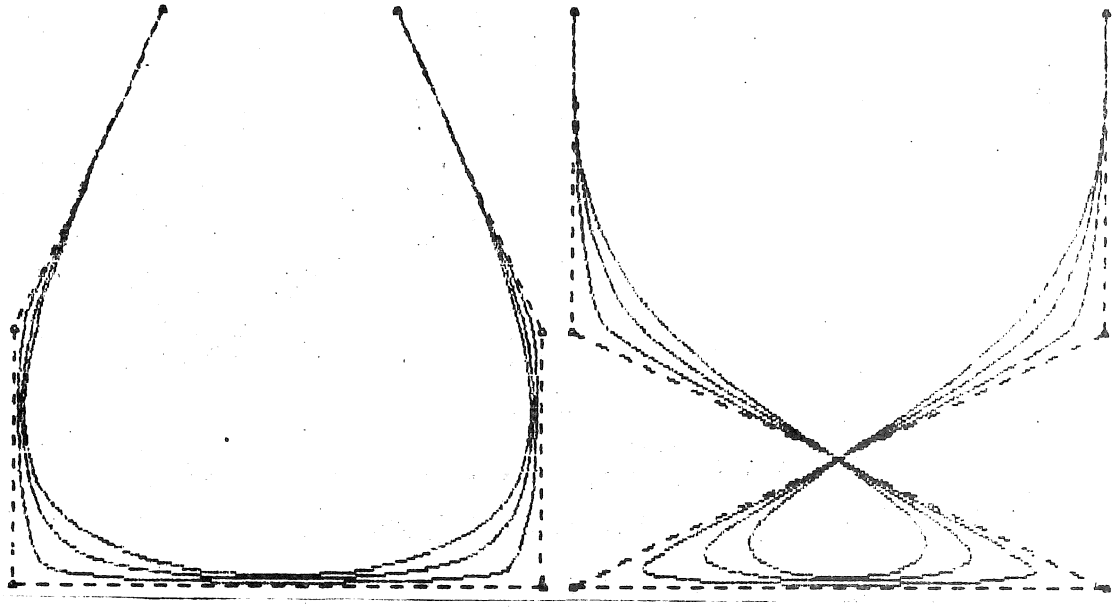
(F) $$B_1^2 \cdot P''_{i-1}(u_i) + B_2 \cdot P'_{i-1}(u_i) = P''_i(u_i)$$

La condición (D) establece continuidad en la posición. La condición (E) permite vectores tangentes colineales pero con una discontinuidad en su magnitud (B1) llamado 'Sesgo' (Bias). Físicamente ésto representa un cambio instantáneo de la velocidad de la curva en la misma dirección en la unión entre dos segmentos. La condición (F) preserva la continuidad del vector curvatura en la unión entre dos segmentos, pero con el agregado de un factor B2 llamado 'Tensión' que introduce una componente extra de aceleración en la dirección del vector tangente. Hay también un término de aceleración colineal al vector tangente inevitable debido al B1.

Como puede verse, de esta definición se obtienen dos grados más de libertad cuyo efecto es el siguiente: Si el sesgo (bias) es uno y la tensión es cero, las ecuaciones (D), (E) y (F) se igualan a las (A) y (B) y el método se comporta como el B-Spline. Luego el

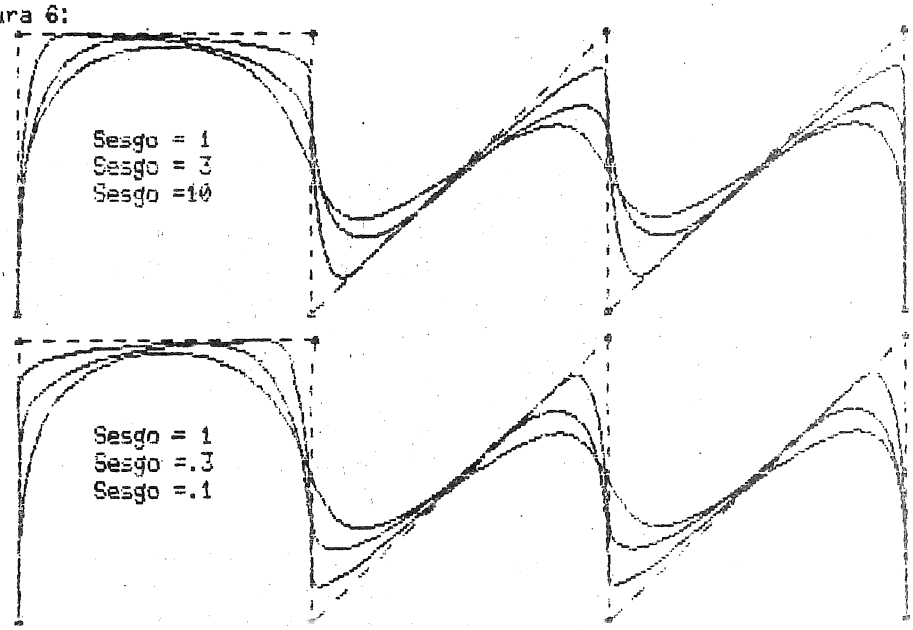
B-Spline es un caso particular del Beta-Spline. En la figura 5 podemos apreciar el comportamiento del Beta-Spline con los polígonos de control anteriores y para tres valores de sesgo y tensión distintos.

Figura 5: Beta-Spline



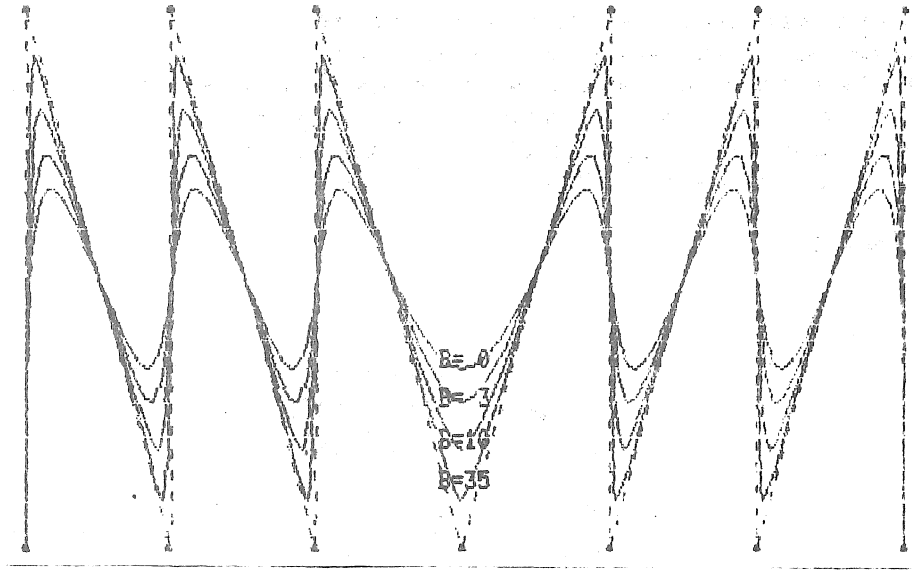
Es de notar cómo los resultados producidos son aún mejores que aquellos obtenidos con el B-Spline. Una explicación más detallada del funcionamiento de los parámetros muestra lo siguiente: en la figura 6 observamos el efecto de variar el sesgo (B1) manteniendo la tensión (B2) en cero.

Figura 6:



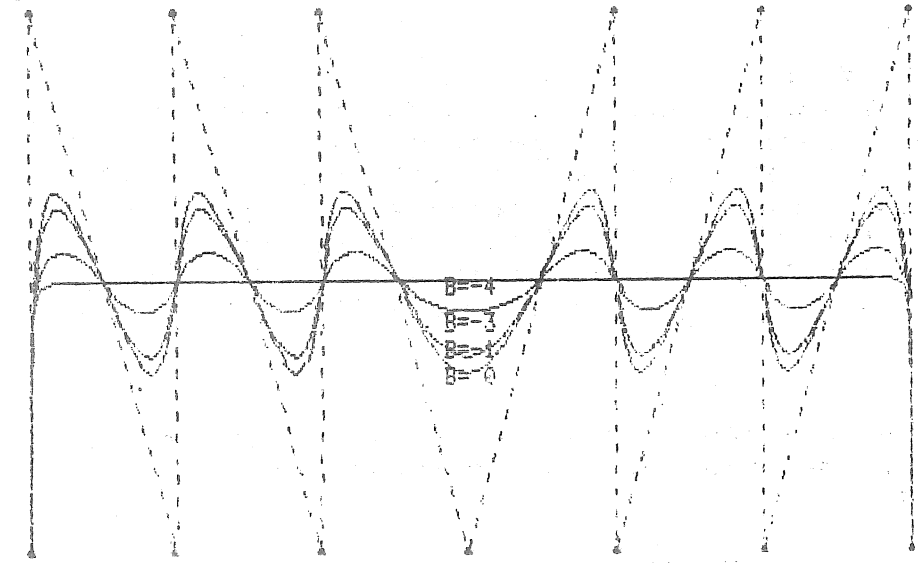
Para valores mayores que uno, la curva se desplaza hacia la izquierda aproximándose más y más al polígono de control. Para los valores recíprocos respecto de la unidad la curva se desplaza hacia la derecha con un comportamiento similar. En la figura 7 se muestra cómo se comporta la curva para distintos valores positivos de la tensión pero manteniendo el sesgo en su valor unitario.

Figura 7:



A medida que aumenta dicha tensión la curva se junta más y más con el polígono de control. En la figura 8 vemos el resultado de aplicar tensiones negativas. Se puede apreciar cómo la curva va separándose del polígono de control hasta prácticamente convertirse en una línea recta.

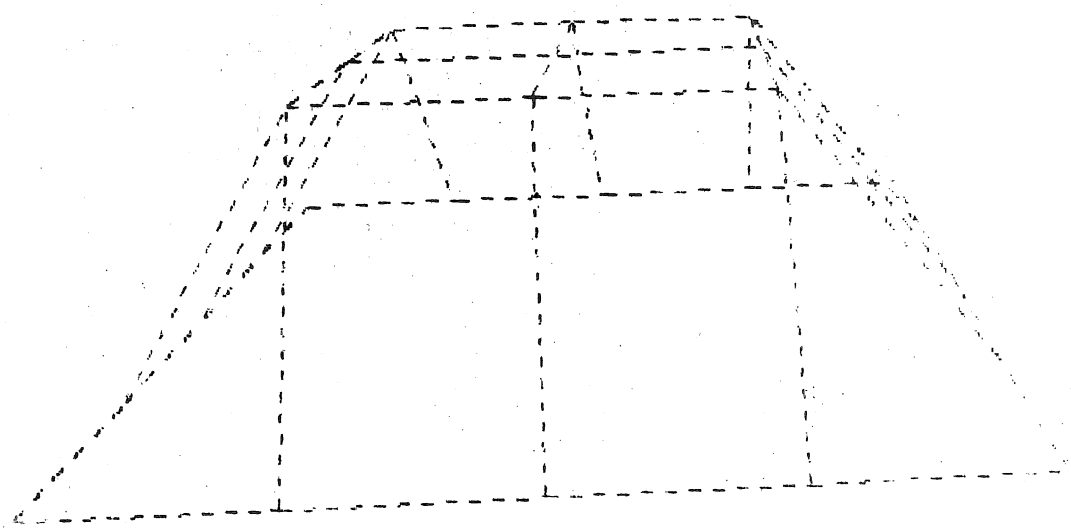
Figura 8:



VI - EXTENSION DEL METODO A LA APROXIMACION DE SUPERFICIES

Para representar una superficie a través de puntos de control se trabajó en un modelo en el cual se han obtenido las alturas Z además de las coordenadas X e Y de cada punto de control. La unión de dichos puntos de control con líneas rectas a X constante e Y constante conforma el polígono de control de la superficie a interpolar, como se muestra en la figura 9.

Figura 9:



La función que aproxima dicha superficie debe ser una función de dos variables que para cualquier coordenada constante traze una curva que cumpla con las propiedades de aproximación de curvas ya explicadas. El método matemático que permite obtener dichas propiedades a partir de los métodos de aproximación de curvas se denomina PRODUCTO CARTESIANO de las funciones. Utilizando dicha herramienta matemática es posible extender la aplicación de los métodos ya vistos al caso general de los puntos de control en el espacio.

La ecuación de una superficie aproximada por el producto cartesiano de uno de los métodos, por ejemplo el Bezier, queda, entonces:

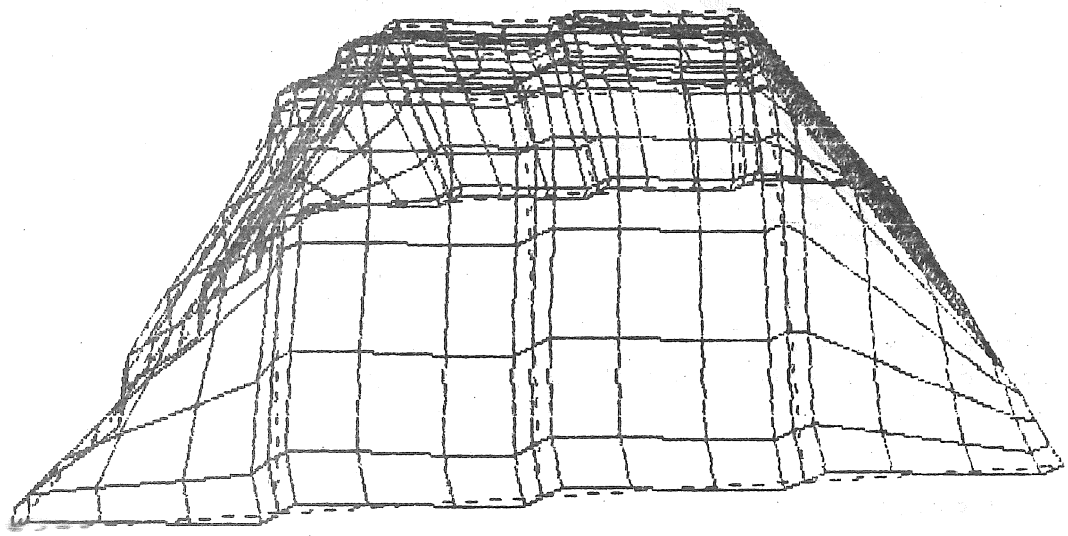
$$P(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,n}(u) \cdot B_{j,m}(v) \cdot P_{i,j}$$

donde N: Cantidad de puntos de control en la coordenada X.
M: Cantidad de puntos de control en la coordenada Y.

u: Parámetro de la superficie, recorrida según el eje X.
v: Parámetro de la superficie, recorrida según el eje Y.
$P_{i,j}$: Valor de la coordenadas (x,y,z) del punto de control i,j.
$B_{i,n}(u)$: Función de mezcla del parámetro u.
$B_{j,m}(v)$: Función de mezcla del parámetro v.
(tal como fueron definidas en el capítulo III)

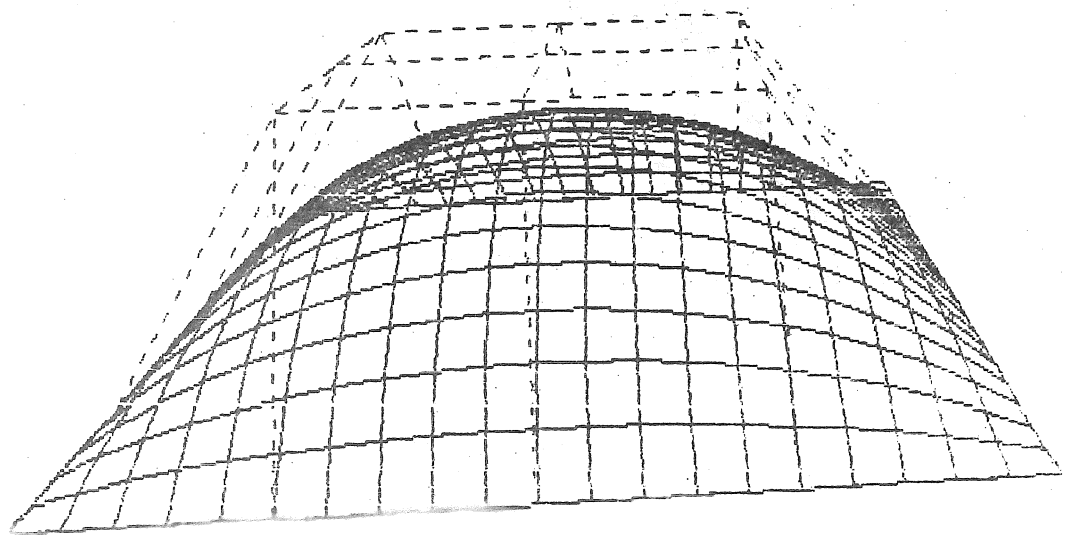
El tratamiento para los otros métodos es enteramente similar. En la fisura 10 se observa el comportamiento del algoritmo Hermite para el grafo de control de la fisura 9.

Figura 10:



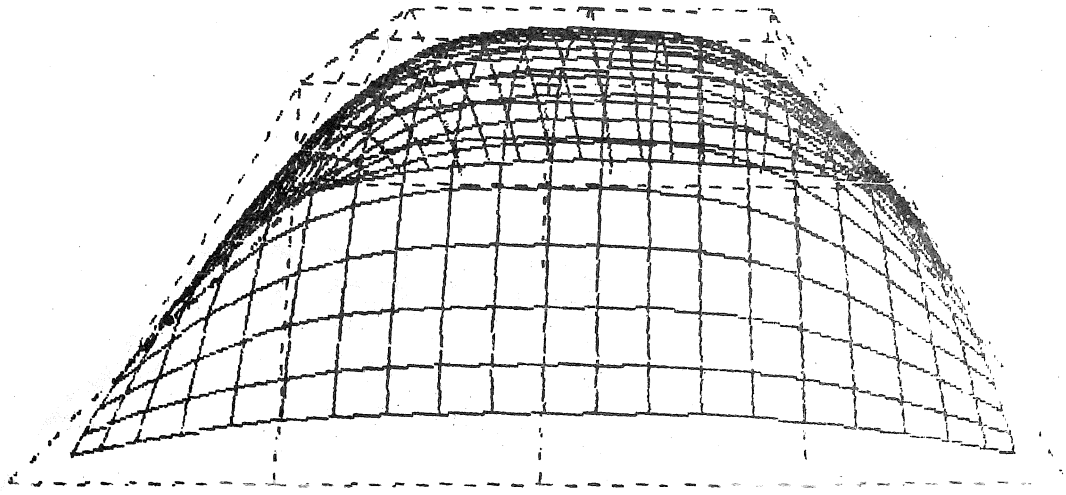
En la fisura 11 vemos el mismo grafo aproximado por Bezier.

Figura 11:



La aproximación realizada por el método B-Spline se observa en la fisura 12.

Figura 12:



Por último se ve el resultado producido para tres valores distintos de sesgo y tensión con el algoritmo Beta-Spline en las figuras 13 y 14

Figura 13:

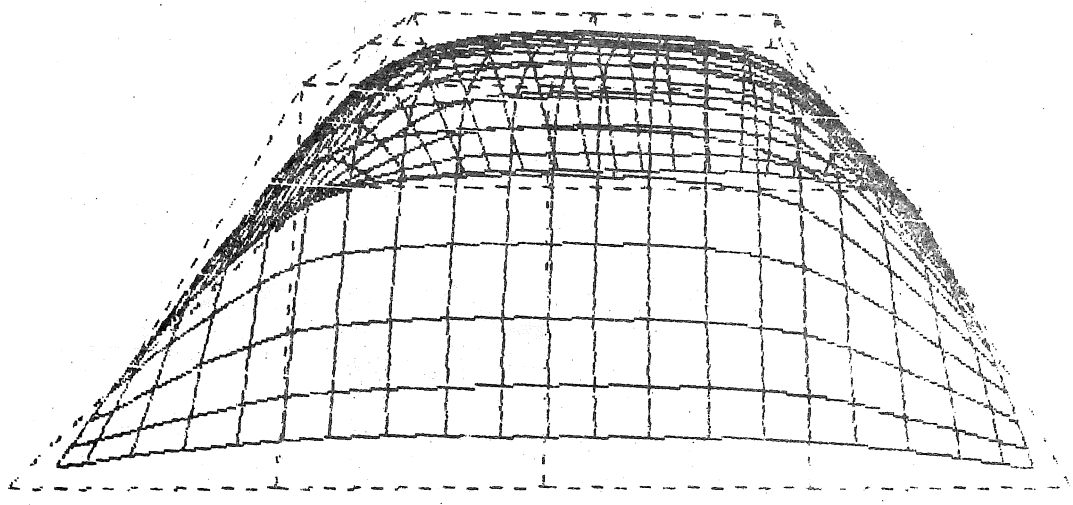
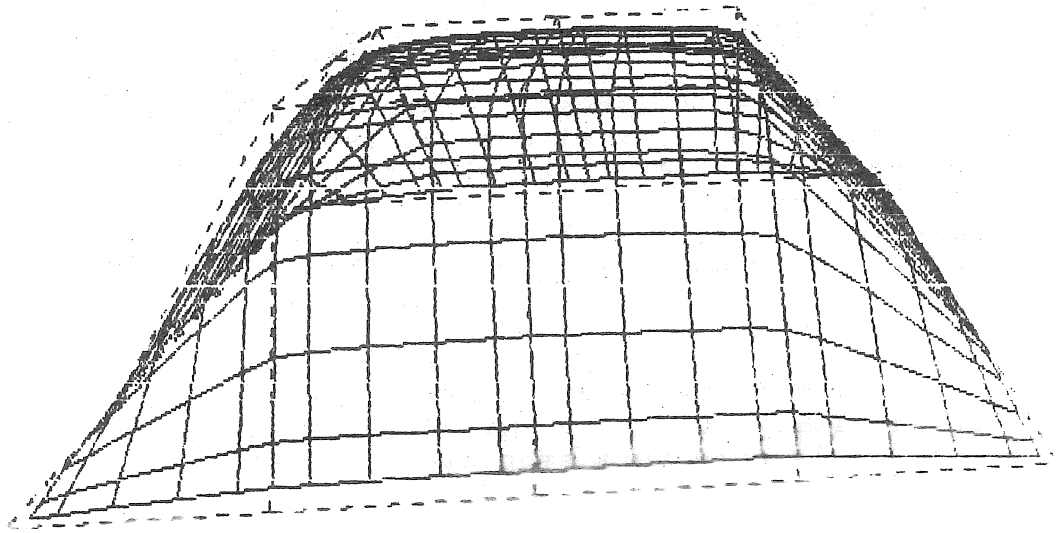


Figura 14:



VII - COMPARACION DE LOS METODOS Y CONCLUSIONES

De la evaluación del comportamiento de los métodos analizados y sus respuestas al conjunto de polígonos de control en el plano y en el espacio surgen las siguientes consideraciones:

De los gráficos de la figura 2 producidos por el método de Hermite se ve cómo, si bien se respeta la forma aproximada del polígono de control, existe sin embargo una discrepancia que acentúa exageradamente las discontinuidades del mismo, siendo el resultado general bastante poco exacto.

En la figura 3 el resultado producido con el algoritmo de Bezier para los mismos polígonos de control muestra que el comportamiento en general es más satisfactorio, aunque se ve cómo para el segundo polígono el método es incapaz de seguir la 'vuelta' dada por el mismo.

Un mejor comportamiento en este caso se observa en la figura 4 en las curvas producidas con el B-Spline. El seguimiento de los polígonos de control es mucho mejor, aún en la 'vuelta' dada por el segundo de los mismos.

El resultado es muy superior con el algoritmo Beta-Spline mostrado en la figura 8. Se produjeron tres de las curvas que es posible obtener con el método gracias a sus grados de libertad adicionales (sesgo y tensión). Es notable cómo gracias a ello podemos hacer parecerse las curvas trazadas al polígono de control tanto como se desee.

En la figura 10 vemos en trazo discontinuo el polígono de control de la figura 9 el cual es aproximado con el método de los polinomios de Hermite (en trazo lleno) con la extensión para figuras en tres dimensiones explicado en VI. Se observa que el comportamiento es similar al de la figura 2 en el sentido de presentar 'curvaturas' ajenas al polígono de control que lo alejan del realismo.

En la figura 11, en el gráfico producido por el método de Bezier en el mismo caso, el resultado es más armónico que el anterior aunque tal vez la curvatura producida siga escasamente al polígono de control.

En la figura 12, en cambio, observamos que esta curvatura es seguida con más fidelidad por el algoritmo B-Spline. El mejor comportamiento se observa al notar que la curva alcanza a tocar el polígono de control en la parte superior del mismo, cosa que no sucedía con el método de Bezier.

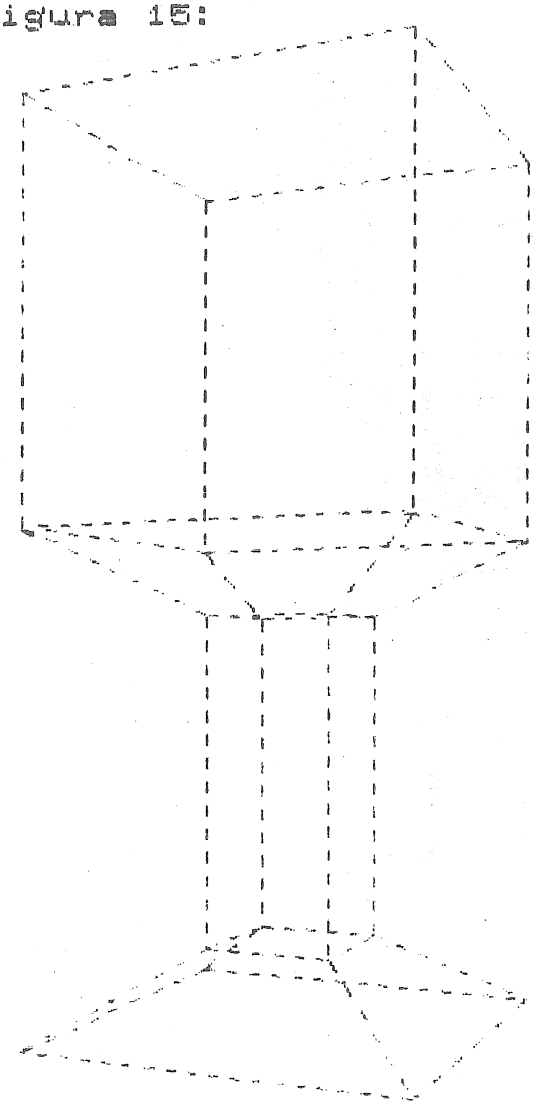
En la figura 13 vemos una de las posibilidades que nos brinda el Beta-Spline: se tensionó el gráfico de modo que se ve cómo se ha acercado más aun al polígono de control que en el B-Spline. Por último en la figura 14 vemos otro caso donde se llevó el gráfico más cerca aún del polígono de control.

De todos los resultados mostrados surge claramente que el método Beta-Spline es al mismo tiempo el más versatil y el que produce los resultados gráficos más realistas. Su rapidez es comparable a la de los otros algoritmos siendo superior la calidad gráfica producida, independientemente del dispositivo utilizado y aún en terminales gráficos de baja resolución. Por dichas razones se adoptó este método como el utilizado para realizar la parte siguiente del presente trabajo.

VIII - TRABAJO DE APLICACION

Como trabajo de aplicación se elisió el modelo de una copa cuyo polígono de control, mostrado en la figura 15, está construído con una base de datos de tan solo 20 puntos.

Figura 15:



En la figura 16 se observa la copa dibujada con línea llena a partir de los puntos de control (dibujados con línea de trazos). La mencionada versatilidad del método se aprecia entre las diferencias entre esta figura y la figura 17 en donde se han variado localmente el sesgo y la tensión en distintos lugares de la copa, produciéndose formatos totalmente distintos sin alterar la base de datos.

Figura 16:

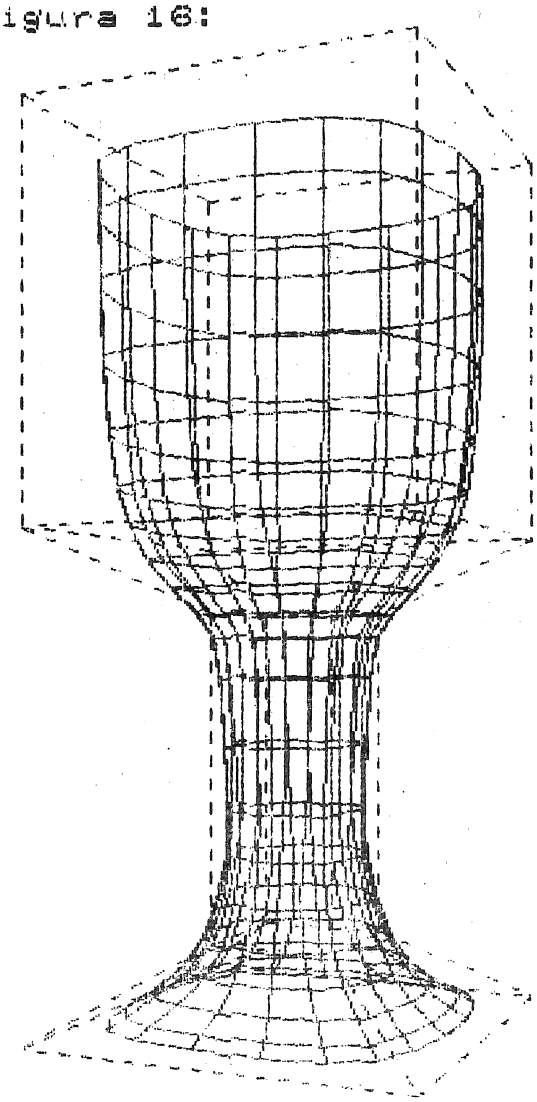
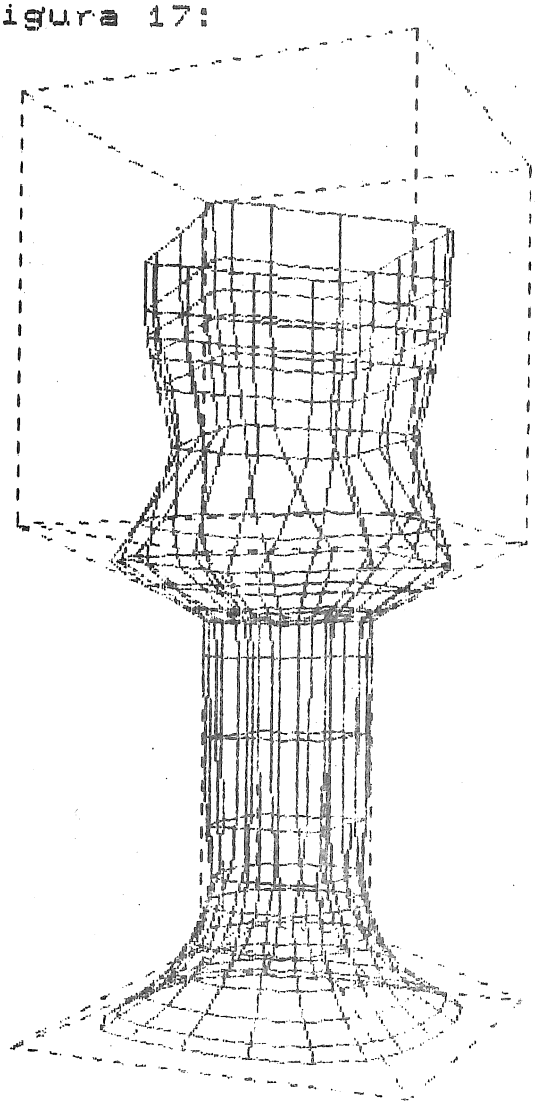


Figura 17:



Para producir resultados de mayor realismo se buscó implementar una rutina de eliminación de líneas ocultas. Dicho algoritmo produce resultados de calidad aceptable y actualmente es objeto de estudio e investigación en este Laboratorio. Con el agregado de dicho algoritmo se produjeron las figuras 18 y 19, donde se aprecian sus resultados combinados con transformaciones tridimensionales.

Figura 18:

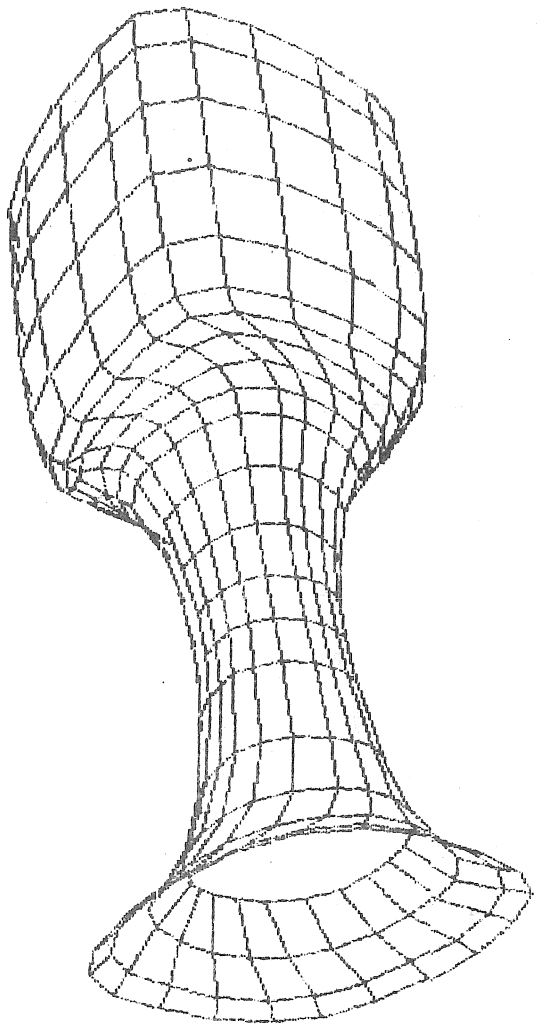
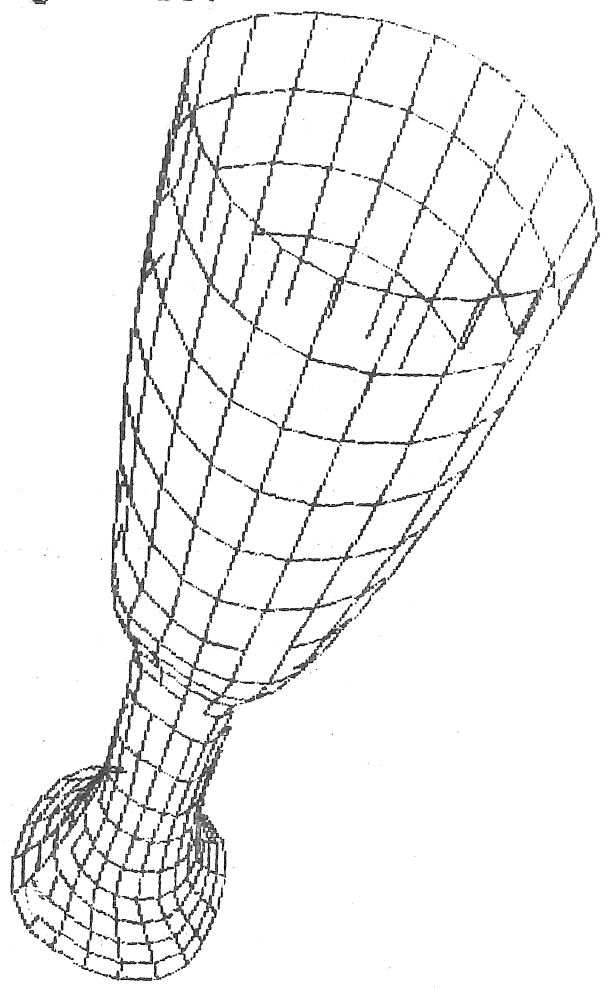


Figura 19:



IX - BIBLIOGRAFIA

[1]: Newmann - Sproull
Principles of Interactive Computer Graphics
Mc Graw-Hill, 1979

[2]: Giloi, W.
Interactive Computer Graphics
Prentice-Hall, 1978

[3]: Foley-Van Dam
Fundamentals of Interactive Computer Graphics
Addison-Wesley, 1982

[4]: Algorithms for Graphics & Image Processing
Theo Pavlidis
Computer Science Press, 1982

[5]: A two-space solution to the hidden-line problem
Thomas Wright
IEEE Transactions on Computers, Jan 1973

[6]: A description and evaluation of various 3D models
Brian Barsky
IEEE Computer Graphics, Jan 1981

[7]: Perspective representation of functions of two variable
Kubert, Szabo & Giulieri
Journal of the ACM vol.15 n.2, Apr 1968

[8]: Local control of Bias and Tension in Beta-Splines
J. Barsky-J.Beatty
Computer Graphics of the ACM vol. 17 n. 3, Jul 1983

[9]: Urn Models & Beta-Splines
Ronald Goldman
IEEE Computer Graphics, Feb 1986

[10]: Implemntación de Rutinas Gráficas Tridimensionales
Claudio Delrieux
XV JAIIO
Sociedad Argentina de Informática e Investigación Operativa
Universidad Nacional del Sur, 1985

TITULO : INTRODUCCION A LA TOMOGRAFIA AXIAL COMPUTARIZADA

AUTOR : ING. JOSE J. RAMOS RODRIGUEZ

DIRECCION : AV. AMERICA No. 2848 Y CUERO Y CAICEDO

QUITO - ECUADOR

R E S U M E N

A través de un método didáctico se aspira reforzar o aclarar algunos de los conceptos y métodos de cálculo involucrados en el conocimiento de la Tomografía Axial Computarizada.

Considerando las limitaciones existentes en el diagnóstico por imágenes que tiene la radiología convencional, se justifican la creación y el vertiginoso desarrollo de este nuevo método de diagnóstico médico.

En una breve reseña histórica se evidencia el grado de evolución alcanzado; y mediante una descripción de las cinco generaciones de tomógrafos, se intenta explicar en forma lógica y práctica todas las técnicas de exploración existentes.

Una vez descritos los fundamentos básicos y las principales partes que conforman un sistema tomográfico, por medio de un modelo, se explican las diferentes fases de la adquisición de datos, procesamiento de la información, presentación y manejo de imágenes.

******

INTRODUCCION

En tecnología radiológica, la absorción posicional de radiación de un objeto es evidenciado por la acción de la energía residual sobre las películas radiográficas, produciéndose la compresión de un volumen en un plano. Por lo tanto, las placas radiográficas son el resultado de elementos de imágen superpuestos que impiden la detección de pequeñas diferencias de absorción. La utilización de medios de contraste incrementa notablemente la capacidad de observación y diagnóstico pero sin permitir una descriminación cuantitativa de las características de absorción y densidad de los tejidos radiados.

La tomografía axial computarizada utiliza los principios tradicionales de la radiología convencional para la obtención de datos de atenuación desde distintas direcciones, de una sección determinada del cuerpo (Fig. 1). Mediante un proceso matemático complejo se reconstruye en la gama de grises, una imagen de la sección transversal explorada, a partir de una matriz de elementos de imagen que representan las diferentes características de absorción a los rayos X de las estructuras radiadas.

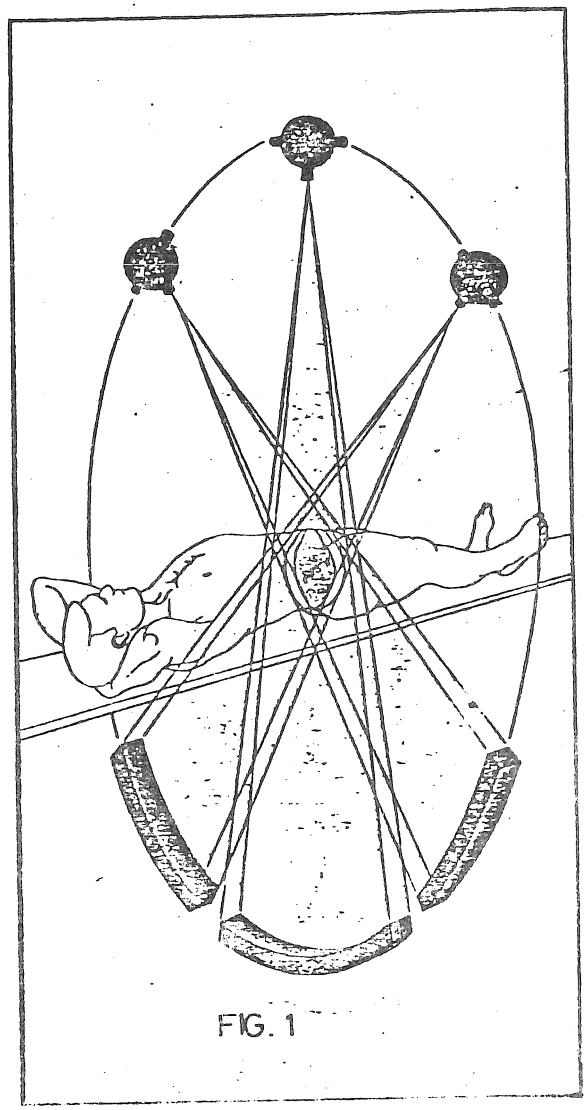


FIG. 1

EVOLUCION DE LA TOMOGRAFIA

El desarrollo vertiginoso de la tomografía axial computarizada puede ser evidenciado en la siguiente descripción histórica y generacional de su proceso evolutivo.

- OLDENDORF en 1961, fue el primero en encontrar una nueva posibilidad de medición de cuerpos de densidades irregulares; llegando a sugerir, un método lógico de procesamiento de la información adquirida, el mismo que fracasó debido a que no contaba en esa época con un medio técnico idóneo para poner en práctica sus ideas.

- CORMACK en 1963, ofreció la primera base teórica para la reproducción de imágenes radiológicas, definiendo inclusive los principios básicos de la tomografía computarizada actual.

- HOUNSFIELD en 1973, construye el primer tomógrafo computarizado aplicado a la medicina. Esta unidad permitía realizar una estimación cuantitativa de la absorción de los rayos X en el cerebro.

- AMBROSE (1973), en un período de dos años, realizó experimentos clínicos con este equipo en el Hospital Morley de Atkinson, perfeccionando el tomógrafo inicial y los métodos de diagnóstico clínico.

- LEDLEY y DI CHIRO, poco tiempo después, desarrollaron el primer tomógrafo computarizado para cuerpo entero, mejorando notablemente la capacidad diagnóstica médica.

Es deseable que un sistema tomográfico proporcione la máxima calidad de imagen, con la mínima dosis de radiación, obtenida con el menor tiempo posible de exploración y procesamiento de información. Con estos ambiciosos fines, se han desarrollado cinco técnicas de exploración o generaciones tomográficas

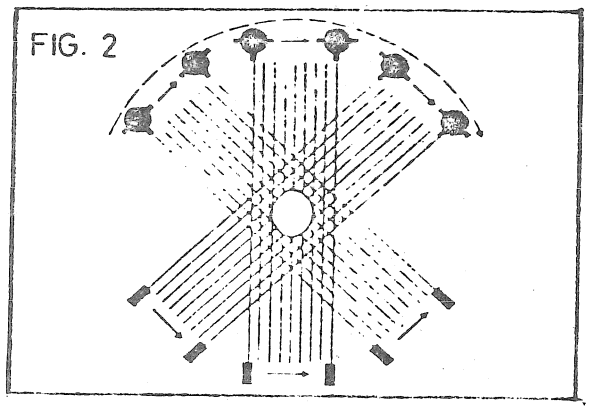
PRIMERA GENERACION (Fig. 2)

Sistemas de Rotación - Traslación con un solo detector.

- El haz de radiación alcanza al único detector situado en posición opuesta al tubo de rayos X.

Para la adquisición de datos el conjunto tubo-dtector realiza un movimiento de traslación lineal a través del cuerpo sujeto a exploración, después realiza un movimiento de rotación, hasta completar un giro de 180°en pasos de 1° Se requerirían aproximadamente 4 minutos para completar un corte tomográfico. La mecánica era sumamente compleja y los tomógrafos estaban diseñados únicamente para cerebro.

FIG. 2



SEGUNDA GENERACION (Fig. 3)

Sistemas de Rotación - Traslación con varios detectores

- El sistema de detección con cámaras de 5 a 50 detectores es colocado opuesto al tubo de rayos X. El haz de radiación en forma de abanico disminuye el número de pasos angulares, los cuales son usualmente de 10°. La adquisición de datos se realiza únicamente durante cada movimiento de traslación lineal y deja de tomar datos durante cada movimiento de rotación, requiriéndose entre 10 y 30 segundos por cada exploración.

  Aparecen los primeros tomógrafos para cuerpo entero; pero mecánicamente siguen siendo complejos.

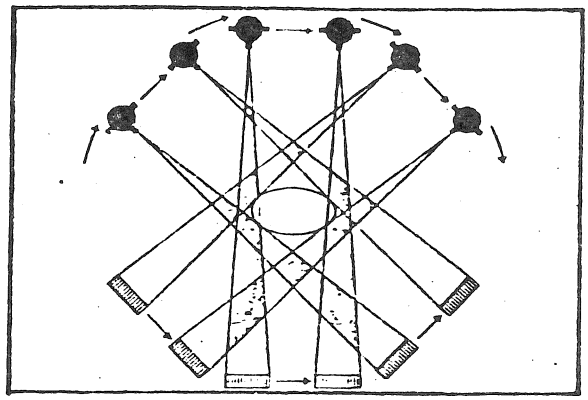


FIG. 3

TERCERA GENERACION (Fig. 4)

Sistemas de Rotación - Rotación

- Un haz de radiación en forma de un amplio abanico cubre completamente una cámara de 200 a 1000 detectores los cuales se mueven en el mismo sentido del tubo de rayos X alrededor del cuerpo explorado.

  La adquisición de datos se lo realiza en forma pulsante durante su recorrido de 360°. Mecánicamente se simplifican los tomógrafos y se pueden obtener tiempos de exploración que permanecen entre 1 y 10 segundos.

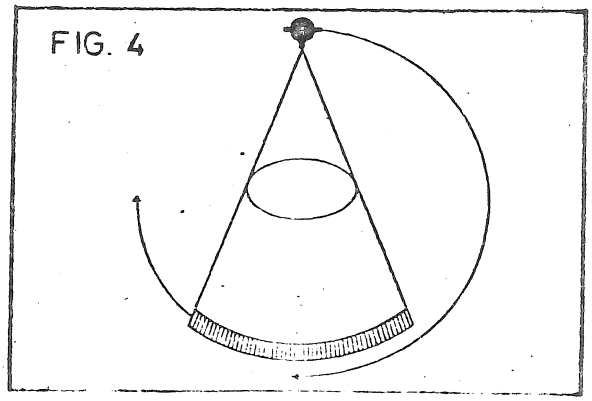


FIG. 4

CUARTA GENERACION (Fig. 5)

Sistema de rotación con detectores estacionarios

- Un tubo de rayos X con un abanico amplio de radiación se mueve angularmente por la parte interior o exterior de un aro fijo de detectores (desde 300 o varios miles de detectores).

  La adquisición de datos se la realiza en forma pulsante durante 3 a 8 segundos.

  El gran número de detectores encarece el sistema y únicamente una porción de ellos es usado en un tiempo dado. El gran espaciamiento entre detectores disminuye su eficiencia.

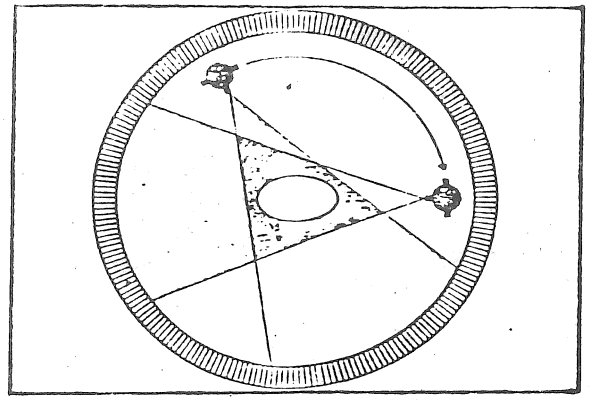


FIG. 5

QUINTA GENERACION

Sistema estacionario

- Varios tubos de rayos X con abanicos amplios de radiación, son colocados concéntricamente a un aro fijo de varios miles de detectores. Cada tubo de rayos X sensibiliza el grupo de detectores opuestos a él. Este sistema por su alto costo de producción, en la actualidad no es factible comercialmente y se encuentra en una fase investigativa. La adquisición de datos puede realizar en tiempos menores que un segundo y se aspira obtener imágenes prácticamente en tiempo real.

Una breve descripción de algunos de los conceptos involucrados en la tomografía axial computarizada permitirán una evaluación del sistema.

Absorción : Es la capacidad, asociada con la transferencia de energía, de un material para detener los rayos X.

Intensidad incidente : Es el número original de fotones de rayos X que ingresan a un objeto.

Intensidad transmitida : Es el número de fotones de rayos X que pasan a través de un objeto.

"Scatter" : Es la dispersión de fotones de rayos X.

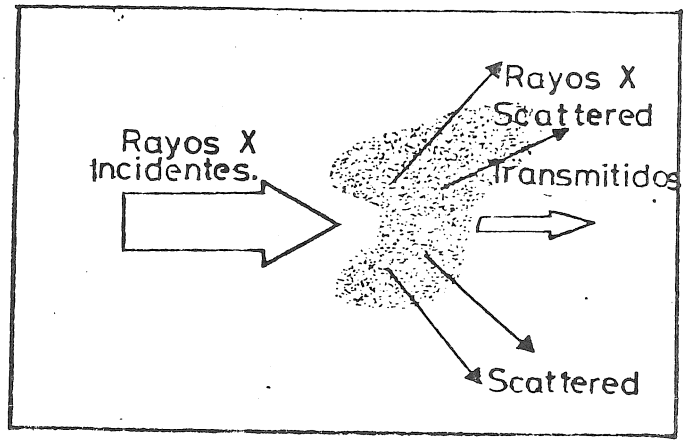


FIG. 6

Atenuación : Es la reducción de intensidad de la radiación que pasa a través de un material debido a la absorción y al "Scatter".

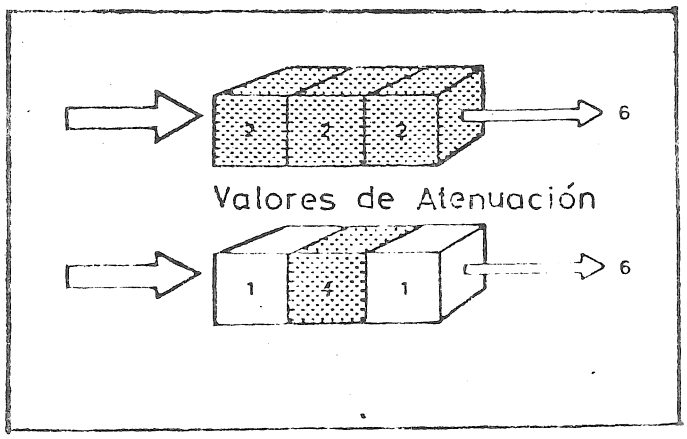
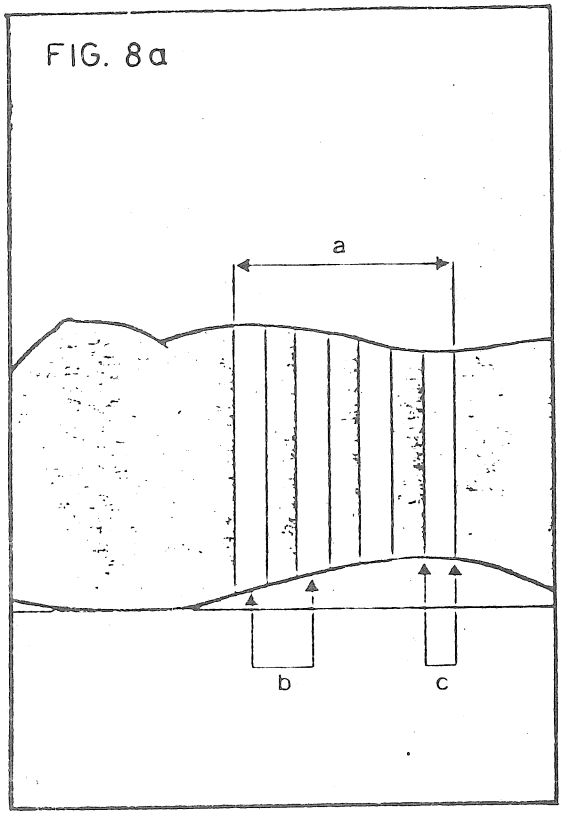


FIG. 7

Corte Tomográfico : Es la sección transversal del cuerpo, de grosor determinado por el haz de radiación, que proporciona los datos de atenuación para la reconstrucción de la imagen.



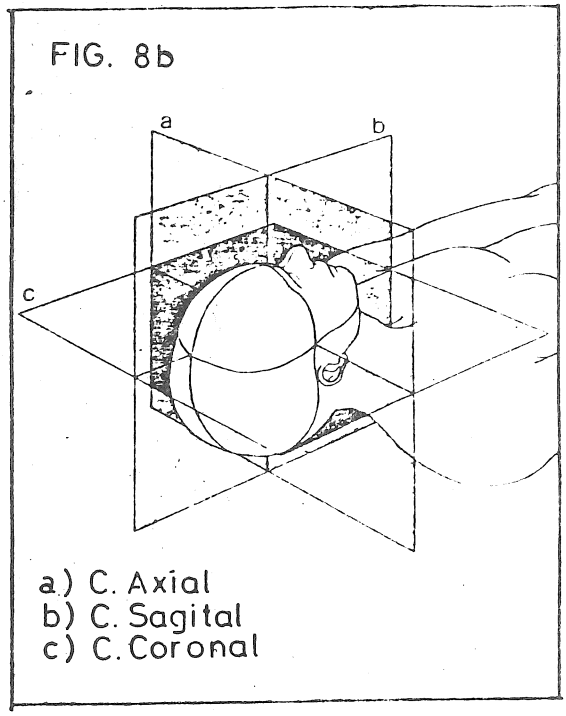


Fig. 8a a) Area de exploración; b) intervalo de corte; c) ancho del corte tomográfico.

Matriz de imagen : Es un arreglo bi-dimensional de valores que representan elementos de imagen en esa localidad.

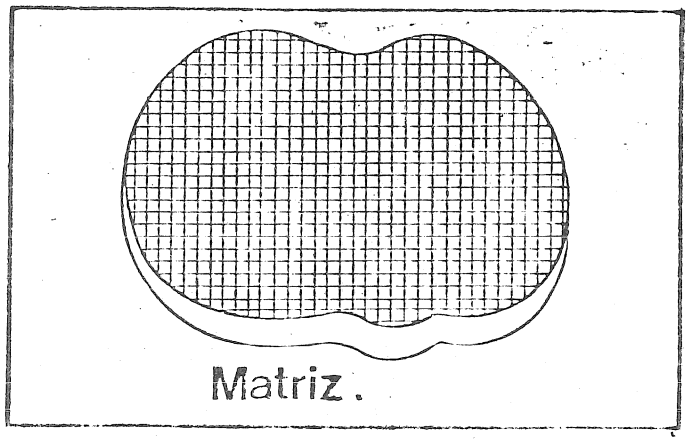


FIG. 9

Voxel : Es cada elemento individual de volumen en el objeto explorado.

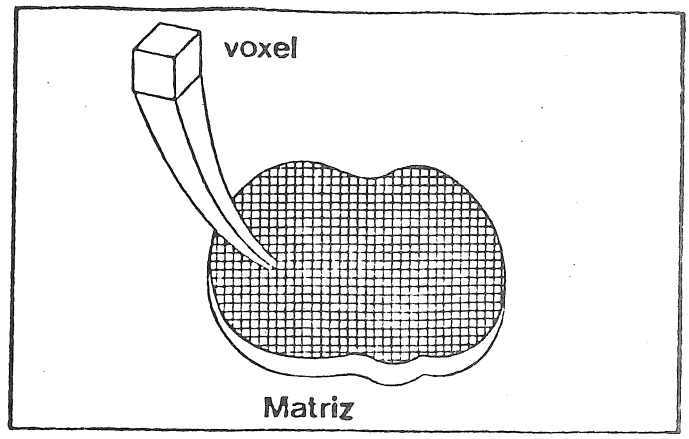


FIG. 10

Pixel : Es cada elemento individual de imagen.

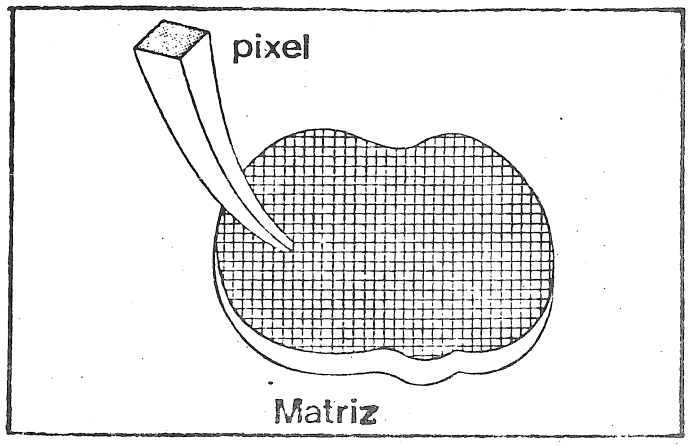


FIG. 11

Número CT o Hounsfield : Son los números de la matriz de imagen que representan el valor de la atenuación en el correspondiente voxel del objeto.

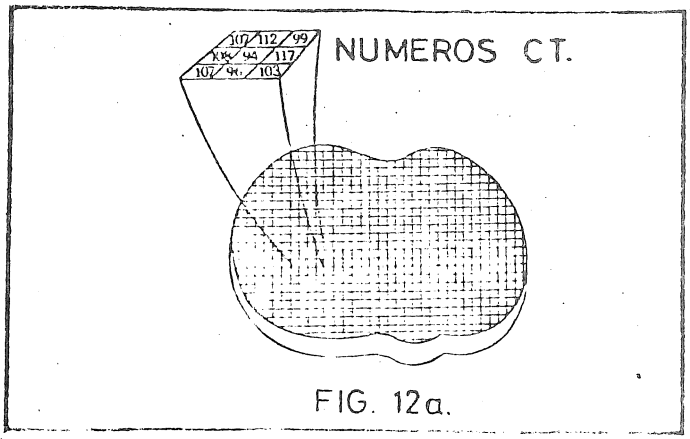


FIG. 12a.

Un tomógrafo axial computarizado clásico consta de las siguientes partes básicas (Fig. 12)

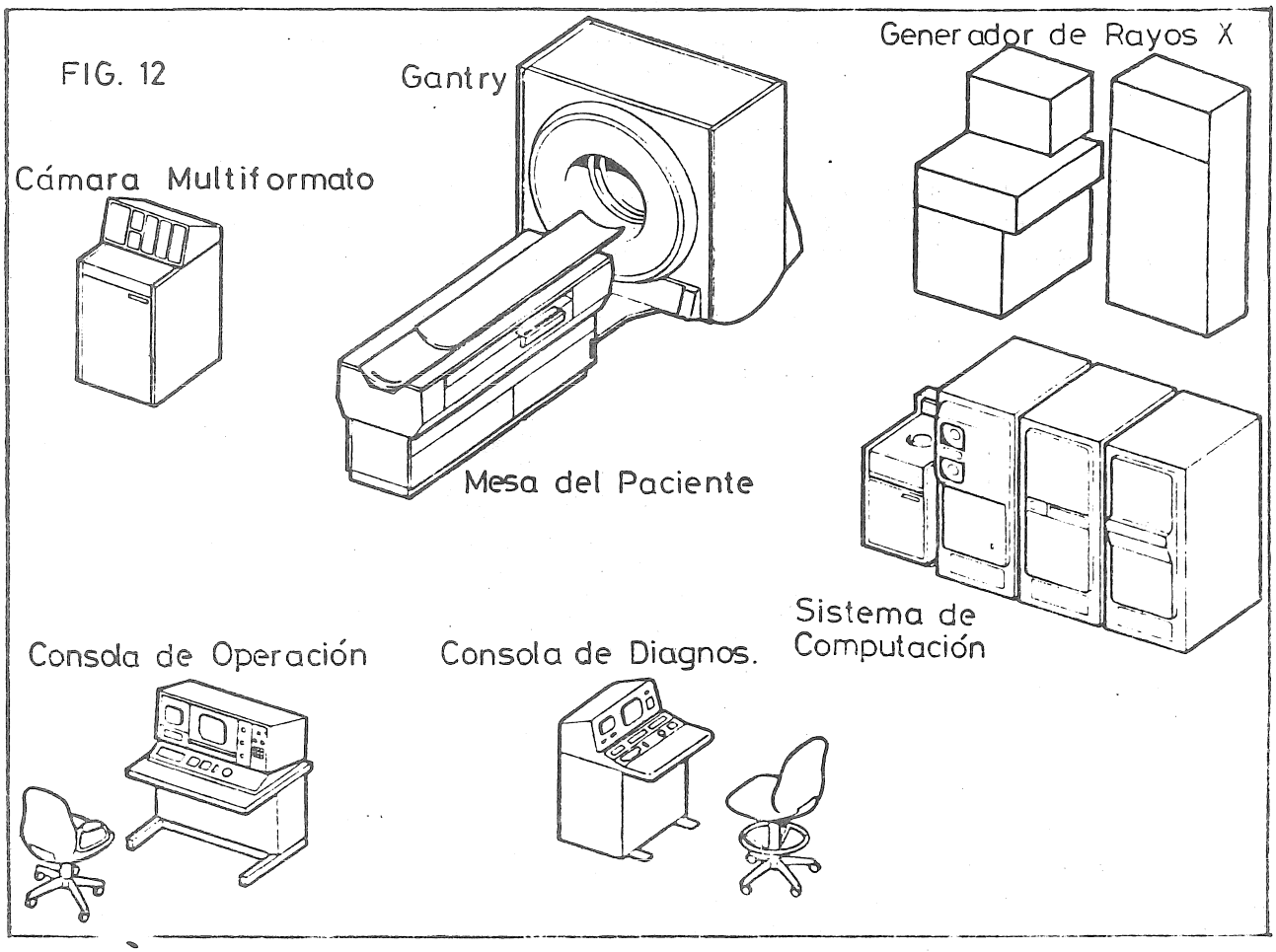
- El computador, es la unidad encargada de controlar todas las funciones del sistema, procesar la información y reconstruir las imágenes.

- La unidad de exploración o gantry, consistente de un tubo de rayos X, un sistema de detección y adquisición de datos.

- El control y generador de rayos X, es la unidad de potencia encargada de producir la energía necesaria (120KV - 500 mA) para la producción de rayos X.

- La consola de operación y diagnóstico, es la unidad que permite el diálogo con el computador y la evaluación de la imagen procesada.

- Las unidades de almacenamiento de información, permiten guardar temporal o permanentemente las imágenes de pacientes, y éstas pueden ser discos magnéticos, ópticos, floppies, cintas magnéticas, placas radiográficas, fotografías, etc.



El sistema de rotación - traslación de un solo detector servirá para explicar las fases de adquisición de datos, reconstrucción y presentación de imágenes.

Estos conceptos pueden ser extensibles a las demás geometrías existentes.

Un as colimado de rayos X atravieza un objeto y durante el primer recorrido transversal, sensibiliza al detector que capta los valores de atenuación de la primera vista (Fig. 13).

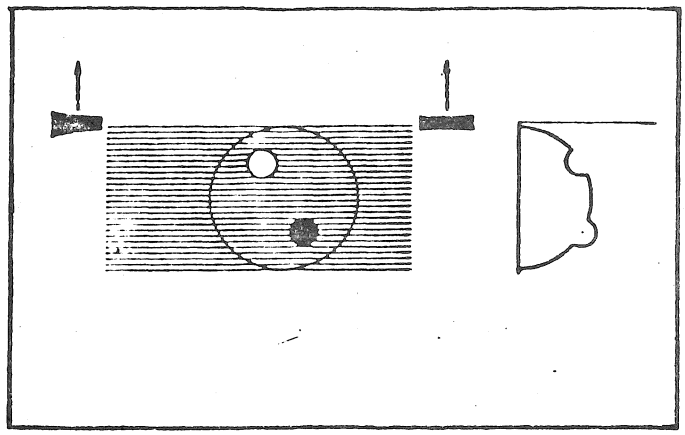


FIG. 13

Después de que la primera vista es grabada en un perfil de atenuación, el - conjunto tubo-detector gira un pequeño ángulo y el proceso se repite hasta obtener muchas vistas del corte tomográfico. (Fig. 14).

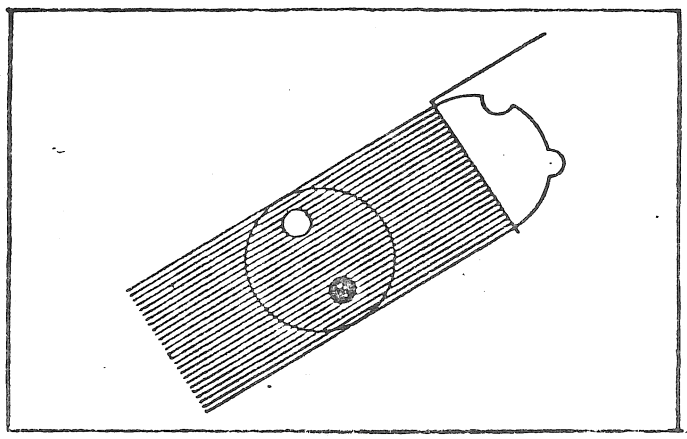


FIG. 14

De cada uno de los perfiles de atenuación o vista, se puede obtener una imagen representativa del objeto explorado. Una imagen puede ser reconstruída por la retroproyección de muchas vistas. (Fig. 15).

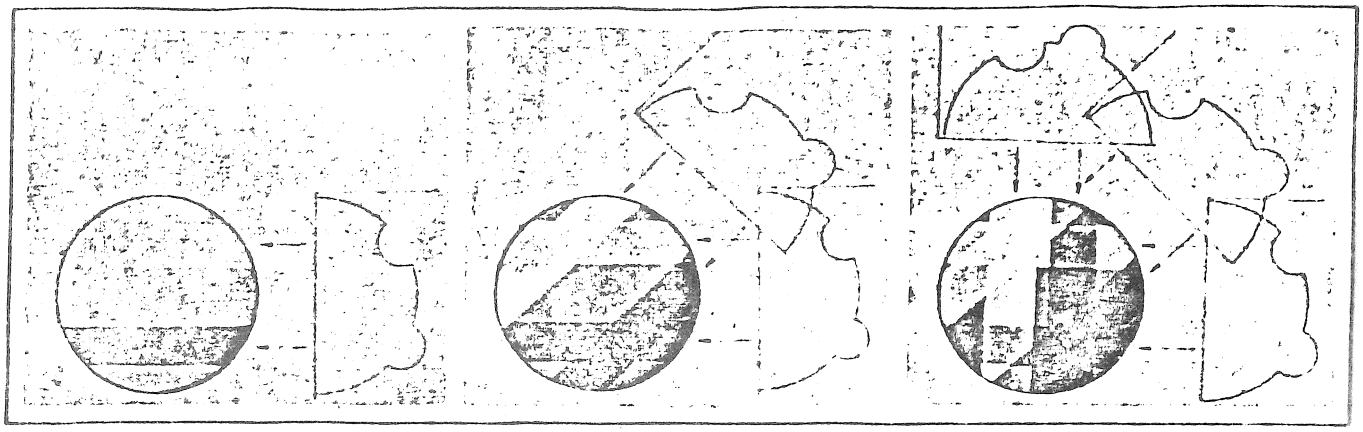


FIG. 15

Los elementos de la imagen reconstruída tendrán una forma de estrella, o con artefactos que pueden ser minimizados mediante la introducción de funciones filtro, las cuales son superpuestas a los perfiles de atenuación por medio de un proceso de convolución.

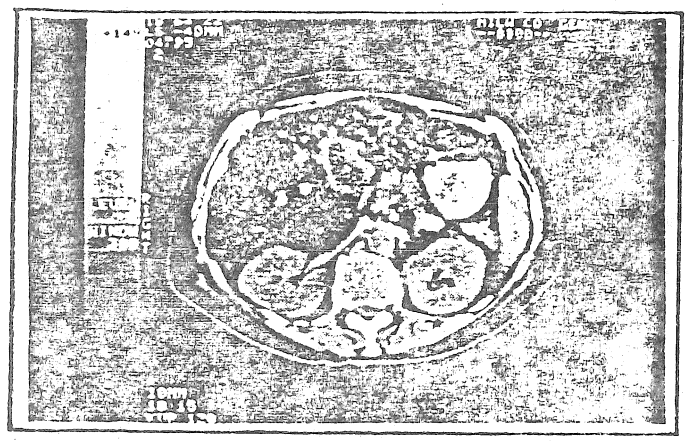


FIG. 16

La influencia de parámetros ajenos al cuerpo explorado como son el estado actual del tubo y detectores, filtros mecánicos, etc; son considerados en un archivo de calibración que corrigen los perfiles de atenuación. Esta información filtrada y corregida es almacenada como valores numéricos en función de su posición, que luego de ser procesada vienen a constituirse en la matriz de imagen, que puede ser presentada en un monitor de televisión para su interpretación.

Cada elemento de imagen o pixel, es el resultado del promedio de todos los valores de atenuación en esa posición y es representado por un tono en la gama de grises para su presentación. Los tamaños más comunes de las matrices de imagen son 256 x 256, 320 x 320, 512 x 512 y corresponde al número de pixels

que contiene la imagen del cuerpo explorado.

El amplio rango de valores de atenuación existentes en una imagen tomográfica requiere de una amplia escala de números CT, los más comunes son de ± 1.000 y de - 1.000 a 3.000. (Fig. 17)

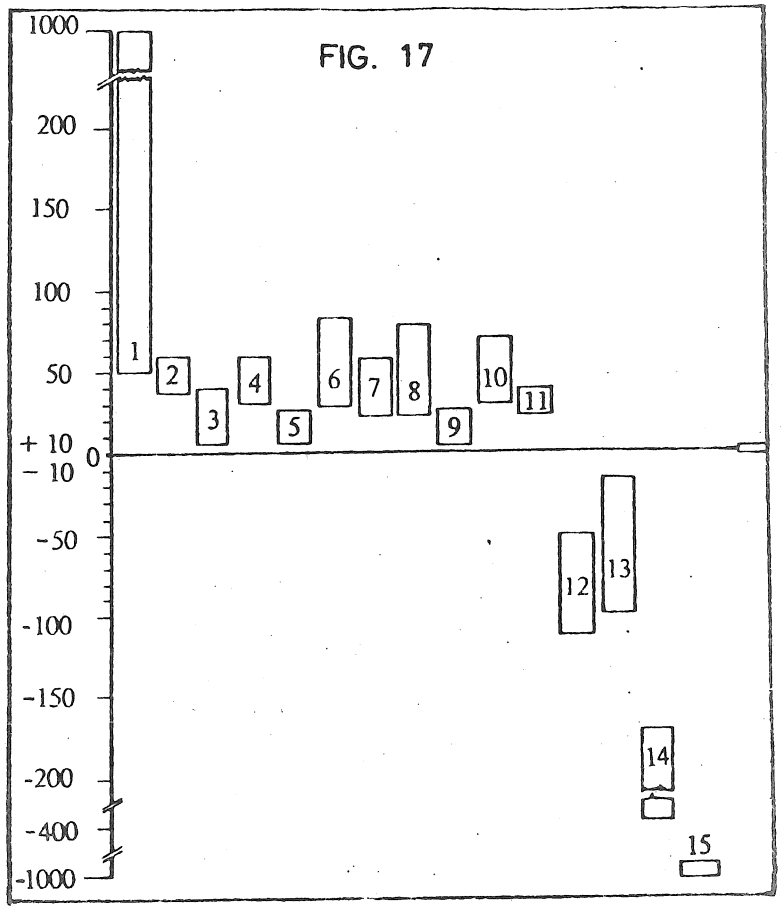


FIG. 17

Sin embargo, el ojo humano no alcanza a distinguir más de 20 niveles de grises y no podría apreciar la gran cantidad de información contenida en una exploración tomográfica. (Fig. 18)

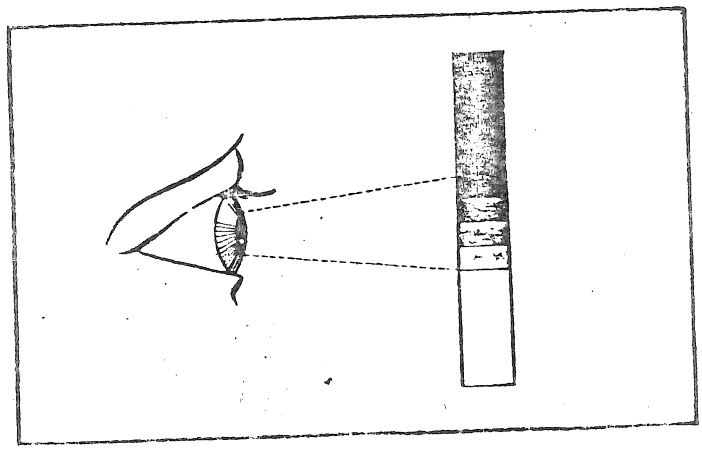


FIG. 18

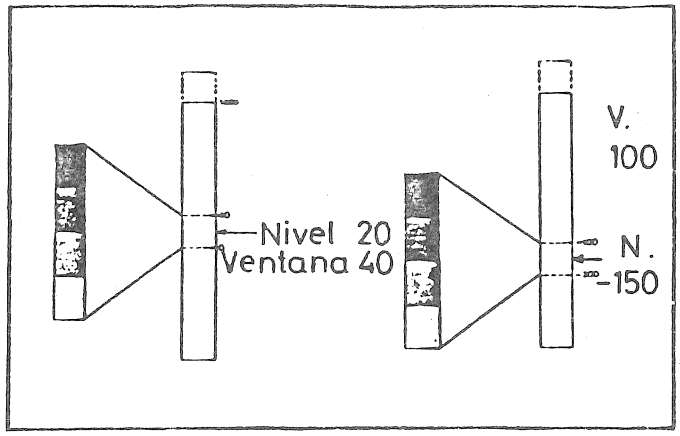


FIG. 19

Dependiendo de la patología que se desea analizar, el médico puede desplazar el nivel de números CT a lo largo de toda la escala, con el ancho de ventana que proporcione la mejor visualización. Manipulando el nivel y las ventanas se puede conseguir que unicamente el rango de densidades escogido, aparezca en la gama de grises completa, mientras la información innecesaria para el diagnóstico es eliminada.

La tomografía axial computarizada se ha desarrollado notablemente, revolucionando las técnicas tradicionales de diagnóstico por imágenes, sin embargo, aún queda mucho por hacer en esta area...

SISTOM - Um Sistema Para Visualização Tridimensional de Tomografia Computadorizada

Adriano Ramos Gonçalves
Karen Basso
Paulo Roberto Partichelli
Jacob Scharcanski
Magali T. Longhi

INSTITUTO DE INFORMÁTICA
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
AV. IPIRANGA, 6681
PORTO ALEGRE - RS - BRASIL

Este trabalho apresenta o projeto SISTOM, atualmente em curso no Departamento de Informática da Pontifícia Universidade Católica-RS. Este tem por objetivo gerar imagens tridimensionais para visualização e manipulação a partir de imagens bidimensionais produzidas por tomografia computadorizada. As imagens bidimensionais representam fatias (ou 'slices') de uma determinada região anatômica em estudo.

Para representar esta região por uma imagem tridimensional, tomam-se / diversas imagens bidimensionais produzidas por tomografia computadorizada, que representam amostragens intercaladas da região. Para que se tenha a representação de toda a região observada, criam-se os 'slices' intermediários à partir da interpolação linear dos 'slices' imediatamente superiores e inferiores aos mesmos. Uma vez que a região anatômica está totalmente reproduzida por 'slices' bidimensionais, armazena-se estes 'slices' em uma estrutura de dados do tipo 'Octree'. Esta estrutura de dados, por ser do tipo árvore, permite / uma eficiente visualização e manipulação desta imagem tridimensional. E por/ se tratar de uma estrutura onde os nodos terminais estão totalmente definidos quanto ao seu preenchimento ou não (presença ou não de cor) e representam uma região bem definida do espaço, permite a sobreposição destes 'slices' visando recuperar a distribuição espacial original dos elementos contituíntes da região anatômica. Com fins de utilização de menos área de memória, utiliza-se a forma de representação 'Octree Linear', que é resultante da forma 'Octree', onde só são armazenados os nodos terminais totalmente preenchidos. Após os 7 'slices' estarem armazenados de forma a constituírem uma imagem tridimensional, percorre-se a estrutura de dados 'Octree Linear', exibindo-se os nodos armazenados.

Pretende-se conduzis a implementação do sistema proposto de forma modular e visando o seu transporte para outros ambientes. Numa primeira etapa / dispõe-se de um compilador TURBO-C, um microcomputador compatível com o IBM-PC/XT com 256 Kbytes, para validar os algorítmos propostos. Num futuro próximo pretende-se transportar o sistema para um equipamento de tomografia computadorizada no Hospital Universitário São Lucas da PUC-RS, obtendo assim um produto de concepção nacional, visando a melhoria da qualidade de diagnósticos no Departamento de Radiologia desta instituição.

UTILIZACION COMPARADA DE DIVERSOS ALGORITMOS DE VISUALIZACION DE FACETAS EN C.A.D.

O.Santana, J.B.Albertini, B.Aubry, G.Meunier, J.L.Coulomb

Laboratoire d'Electrotechnique de Grenoble (UA CNRS 355)
ENSIEG, BP.45
38402 St. Martin d'Hères
FRANCIA

En el *Laboratoire d'Electrotechnique de Grenoble* (LEG), trabajamos en la elaboración de un software de cálculo de campos electromagnéticos tridimensionales bajo el método de los elementos finitos. La finalidad de este software es la concepción y la modelización de máquinas eléctricas. La descripción geométrica y el mallaje *(mesh)* de los objetos tratados con este software, necesitan una visualización frecuente bajo diversas posiciones de los objetos. Por esta razón, la rapidez de la visualización es imprescindible.

Para la visualización, cada faceta es considerada como plana y poligonal, de ahí la necesidad de una descomposición de las facetas no planas en sub facetas (por ejemplo las cilíndricas o esféricas).

En el LEG disponemos de pantallas de visualización de tecnologías diferentes : con barrido aleatorio *random-scan display* (Tektronix 4014) y con barrido TV*raster-scan display*. Las pantallas más recientes realizan el llenado de polígonos (Apollo, Tektronix 4105,4107,4205...) ; las menos recientes coloreaban pixel por pixel (Ramtek 6211, Secapa 550,...)

Hemos elaborado algoritmos de eliminación de partes escondidas *(hidden parts)* específicos a cada tipo de pantalla de manera tal de explotar las posibilidades de cada pantalla al máximo. Estos algoritmos se utilizan en bibliotecas de subrutinas fortran-77. Cada software que necesita estos algoritmos, debe inicialmente seleccionar el tipo de pantalla gráfica; esta tarea se hace de forma interactiva gracias a una biblioteca de diálogo.

Los algoritmos realizados son los siguientes:

El algoritmo de ***Z-Buffer*** que consiste en un barrido de la pantalla pixel por pixel de arriba hacia abajo y de izquierda a derecha. En cada pixel una línea perpendicular a la pantalla atravieza el objeto, de esta manera un cierto número de facetas seron intersectadas. La faceta de profundidad máxima es almacenada (el color o la intensidad por ejemplo). El punto de intersección de profundidad máxima (el eje Z apunta hacia el usuario) corresponde a la faceta vista. Este método permite el coloreado de objetos y exige procesadores rápidos.

El algoritmo ***Scan-Line*** barre la pantalla en una serie de planos (XZ) pasando por cada pixel del eje Y. Cada intersección de estos planos con el objeto es descompuesto en una serie de segmentos. Para cada capa de pixels en Y, dibujamos los segmentos después de haberlos clasificado. En relación al algoritmo precedente, éste gana una dimensión.

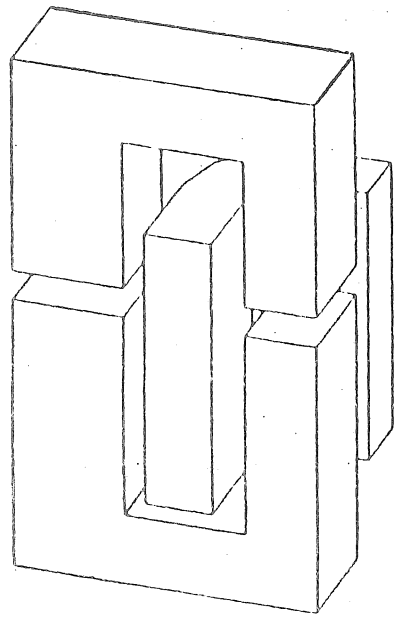
El algoritmo del ***Pintor*** consiste en clasificar las facetas para el dibujo según el orden decreciente de la coordenada Z del centro de gravedad. Este algoritmo es simple y rápido, y da buenos resultados si se trabaja con facetas pequeñas, obtenidas después de un mallaje en elementos finitos, por ejemplo. Este algoritmo funciona solamente en pantallas a borrado selectivo y es fácil de poner en operación.

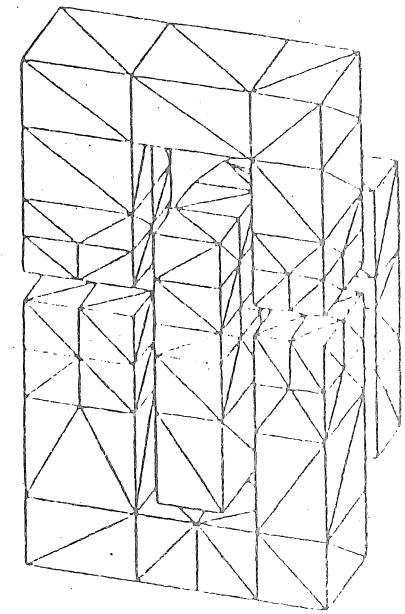
El algoritmo de ***Newell, Newell y Sancha*** funciona igualmente en pantallas a borrado selectivo y llenado de polígonos. El objetivo de este algoritmo es de dibujar sucesivamente las facetas que no

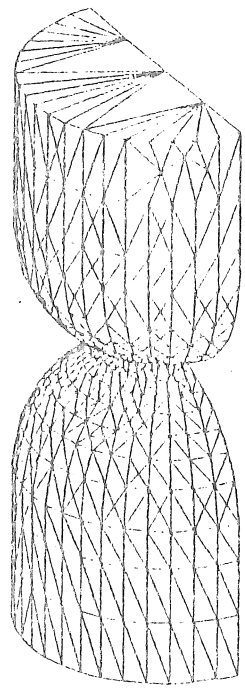
son escondidas o tapadas por las facetas ya dibujadas precedentemente. El problema del grupo de facetas que se cubren mutualmente (por ejemplo A cubre B, B cubre C et C cubre A) es resuelto con el recorte de una o varias facetas del grupo. Gracias a este recorte eventual, el algoritmo resuelve cualquier tipo de ambigüedad.

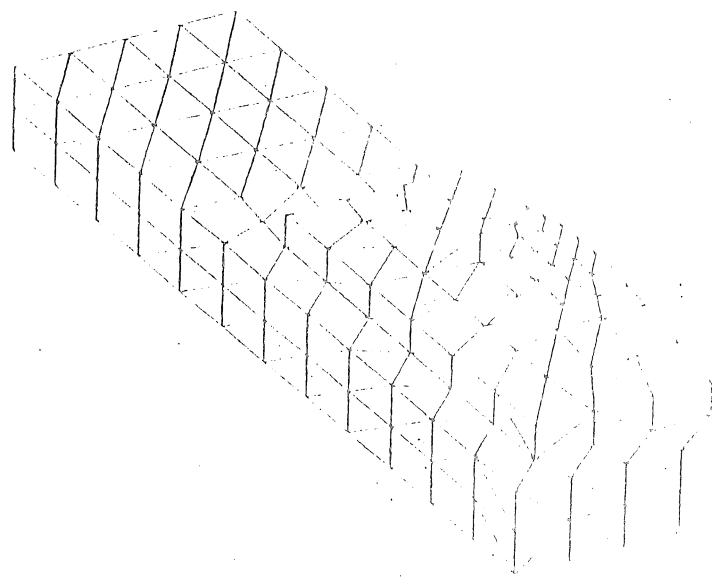
El algoritmo de ***eliminación de líneas escondidas*** calcula en forma exacta los segmentos realmente vistos. Este algoritmo trata solamente facetas triangulares; por lo tanto, es necesario descomponer las facetas iniciales en triángulos (después de un mallaje tetraédrico por ejemplo). Para cada triángulo hay que determinar los pedazos de segmentos que no son tapados por los otros triángulos.

Los algoritmos [1] y [2] no han sido introducidos en el software de CAD *flux3d* elaborado en el equipo de Modelización del LEG dada su lentitud. Los algoritmos [3] y [4] se adaptan mejor a la visualización de facetas planas. Con nuestras pantallas a borrado selectivo utilizamos el algoritmo [4] para visualizar los objetos y el algoritmo [3] para representar el mallaje. Con nuestras antiguas pantallas a borrado no selectivo y con las mesas trazadoras, utilizamos el algoritmo [5] que es mucho más lento.









SISTEMA DE ANIMAÇÃO AUXILIADO PELO COMPUTADOR

Geovanni de Araujo Bertaco
Luiz Fernando Mesquita de Souza
Marcelo Rossoni
Jacob Scharcanski

Instituto de Informática
Pontifícia Universidade católica do Rio Grande do Sul
Av. Ipiranga, 6681 Porto Alegre - RS - Brasil

A Computação Gráfica, especificamente a animação auxiliada pelo computador, é uma das áreas da computação que vem se desenvolvendo intensamente nos últimos anos gerando com isso novos projetos e pesquisas nesta área.

No instituto de informática da PUCRS, esta sendo desenvolvido a nível de pesquisa um sistema gráfico que visa basicamente a criação de objetos facetados tridimensionais e a sua visualização em pespectiva bem como alterações sobre suas estruturas básicas. O sistema interage com o usuário através de uma linguagem de comunicação que além de capacitar o que foi dito anteriormente possibilita ainda gerar-se uma sequência lógica de quadros (cenas) que resultam em uma animação auxiliada pelo computador.

Quanto a animação, e movimentação de cenas ou elementos específicos da cena (atores), o sistema oferece as seguintes operações básicas: Rotação e translação sobre toda uma cena ou em separado para algum ator da cena . Foi também necessário um tratamento especial para as imagens geradas, ou seja elimina-se as chamadas linhas ocultas ("Hiden Lines") excluindo partes da cena que ficam ocultas para o observador. A solução adotada para este sistema foi o método de preenchimento de faces com uma dada cor de preenchimento, onde este preenchimento é feito em uma ordem pré estabelecida para cada face para tanto se classifica todas as faces que fazem parte da cena conforme a sua profundidade em relação ao ponto de observação, antes das mesmas serem exibidas.

A implementação em curso atualmente está baseada em um microcomputador compátivel com IBM PC/XT com capacidade de memória de 256 kbytes utilizando como dispositivo de saída um monitor monocromático e como compilador o Turbo Pascal. Pretende-se em uma etapa posterior dotar o sistema de recursos de realismo tais como sombreamento e textura para a obtenção de imagens de melhor qualidade.

LOS SISTEMAS DE MANUFACTURACION FLEXIBLE (FMS): UNA IMPLEMENTACION DE LA FILOSOFIA DE MANUFACTURACION INTEGRADA POR COMPUTADOR (CIM) A NIVEL DE LA PLANTA.

LIZ JEANNETTE CUBILLOS CANCINO
CRA 74 # 62 - 62 TEL. 2511148

MARTHA LUCIA PARDO MARTINEZ
CRA 74A # 72 - 31 TEL. 2240116

UNIVERSIDAD NACIONAL DE COLOMBIA
FACULTAD DE INGENIERIA
DEPARTAMENTO DE INGENIERIA DE SISTEMAS
Bogotá - Colombia.

Debido al crecimiento en la demanda de bienes, la industria ha tenido que producir en gran escala, implicando la adquisición de máquinas especializadas, personal calificado, grandes inventarios, manejo de materiales y una administración màs eficiente. Todo ello unido a una serie de factores tecnológicos, en el campo de la computación aplicada, y sobre todo en el control y regulación de los sistemas y procesos ha iniciado el camino de la automatización integral de los procesos industriales.

Durante la dècada de los setenta se presentarón algunas situaciones que son los antecedentes a cambios drásticos en la filosofía de la producción, estos son:

- La crisis del petróleo que llevó a optimizar el uso de las máquinas industriales, reduciendo los tiempos muertos en la producción.
- Los sistemas de control de calidad impulsados por los japoneses que tienden a disminuir los inventarios y obligan a producir uniformemente.
- El incremento en la competitividad del mercado que obligó a la mejora en la calidad y la diversificación del producto.

Todo esto unido, llevó a la implantación de técnicas avanzadas de fabricación basadas en un alto grado de automatización con una gran flexibilidad y productividad del sistema.

Además de lo anterior surgierón dos tecnologías, el CAD (Diseño Asistido por Computador) y el CAM (Manufacturación Asistida por Computador), que aunque inicialmente se presentarón como àreas independientes, su evolución y los requerimientos en los procesos de fabricación han llevado a la formación de una tecnología única identificable, el CAD/CAM.

Una verdadera integración de los sistemas CAD/CAM requiere la interacción con otras disciplinas, tales como: CAE (Ingeniería Asistida por Computador), MRP (Planeación de Recursos para la Manufactura), CAS (Simulación Asistida por Computador), CAP (Planeación Asistida por Computador), CAQC (Control de Calidad Asistido por Computador), etc. Cada una de ellas con el objetivo de desarrollar un aspecto neurálgico dentro de un nuevo concepto de producción flexible llamado Manufacturación Integrada por Computador CIM.

La filosofía CIM busca la organización de las actividades de fabricación e información para obtener una solución integrada basada en un sistema de información y control computarizado.

Al igual que en los enfoques tradicionales de manufacturación, el proposito de CIM es transformar materiales y diseños de productos en productos facilmente vendibles a un costo mínimo y en el menor tiempo posible. CIM trabaja sobre la premisa de que la administración debe encaminar sus esfuerzos para optimizar el proceso total del negocio en lugar de las funciones individuales.

Los esfuerzos de CIM van encaminados a la conformación de la fábrica automatizada del futuro, dicha fábrica estará compuesta por FMS (Sistemas de Manufacturación Flexible) que contarán para un desempeño eficiente con: sistemas de manejo de material, máquinas especializadas, robot para manipulación y transporte y un sistema automático de control. Esta fábrica contará con una red jerárquica de computadores que además de controlar los FMS soportará el sistema de Planeación gerencial para la toma de decisiones.

La implementación de CIM exige, además de las disciplinas mencionadas anteriormente, la participación de otras tecnologías, que en esta área se encuentran en un estado primario de desarrollo, entre las cuales se destacan algunas ramas de la Inteligencia Artificial como Robótica, Representación de Imagenes, Lenguaje Natural y Sistemas Expertos. La Tecnología de Grupos y el CAPP (Planeación de Procesos Asistida por Computador) proporcionan la estrategia para una eficiente planeación de la producción.

Los FMS son el resultado de la filosofía CIM en las funciones especificas relacionadas con la sección de manufacturación en la empresa. Debido a esto, los FMS son capaces de producir elementos de alta calidad en pequeños lotes a bajo costo.

Un FMS esta conformado por un conjuto de células de manufacturación, flexibles o nó, cada una de ellas compuesta por una o varias máquinas con CNC (Control Numérico Computarizado) y un robot que sirve a un sistema de manejo de material, todo ello organizado en una red jerárquica computarizada.

El objetivo de un FMS es incorporar muchos conceptos de automatización individual y diferentes tecnologías en un sistema de producción único.

Los Beneficios reportados de la aplicación de los FMS incluyen economias en labor directa e indirecta, flexibilidad en la producción, reducción de inventarios, tiempos menores y mejoras en la calidad del producto y en el ambiente de trabajo. Sin embargo, el cambio de un enfoque de manufacturación convencional a un enfoque de FMS necesita cambios en la planta física, probablemente cambios en los requerimientos de equipo, mayores cambios en la producción, funciones de costo, computación y en la actitud de las personas.

DEFINICION DE LA BASE DE CONOCIMIENTOS PARA UN SISTEMA DE DIAGNOSTICO DE FALLAS EN SISTEMAS DE CONTROL AUTOMATICO DE PROCESOS.

DANIEL SBARBARO H. - DANIEL CAMPOS D.
Universidad de Concepción * Concepción - CHILE

El objetivo del trabajo es la definición, diseño e implementación de una herramienta orientada a la especificación del contenido de la base de conocimientos para un sistema orientado al diagnóstico de fallas en sistemas de control automático de procesos. Para ello, se considera la existencia de conocimiento sobre características de componentes e interrelaciones, historial de fallas y reparaciones, relaciones causa efecto asociadas a fallas y cadenas de fallas, entre otros.

El sistema es modelado en base a la definición previa de un conjunto de componentes normalizados, cuyas relaciones de entrada salida, falla-salida y entrada-falla, se encuentran especificadas en una biblioteca de módulos.

La estructura del sistema bajo control se especifica haciendo uso de una representación gráfica en la cual, mediante iconos, se simbolizan los componentes y las lizasones entre ellos.

A las distintas instancias de un mismo tipo de componente normalizado es posible asociar características específicas, tales como causas probables de fallas en el contexto del sistema bajo control.

En base a la información contenida en la biblioteca de módulos, la descripción de la estructura del sistema y las características específicas de ciertos componentes, se genera un conjunto de reglas que representan el comportamiento del sistema y constituyen la base de conocimientos sobre el sistema bajo control.

Dichas reglas, unidas a la información obtenida desde la instrumentación asociada al sistema, permitirán la realización de inferencias tendientes a determinar posibles fallas asociadas a una cierta situación de alarma, así como causas probables de éstas, lo que constituye un soporte al nivel supervisor de un sistema de control automático de procesos, posibilitando, tanto la toma de decisiones en tiempo real, como el análisis posterior de situaciones de falla ocurridas durante la operación de la planta.

Actualmente se está trabajando en el desarrollo de un postprocesador que, en base a la información generada por el módulo que permite la especificación de los componentes, sus características y la estructura del sistema (Style), realice la generación del conjunto de reglas que constituyen la base de conocimientos, esperandose, finalizar su implementación dentro del presente año.

Un sistema basado en conocimiento, con una orientación como la descrita en está comunicación, podría incorporarse en aplicaciones industriales a nivel nacional, interactuando con sistemas de control de fallas existentes, orientados hacia otros niveles de automatización, posibilitando la utilización de información de tipo estadistico en el proceso de análisis de fallas y causas de éstas a nivel de operación.

APLICACIONES NOVEDOSAS

ASPECTOS DE LA INFORMATICA MEDICA EVOLUCION DESARROLLO Y PERSPECTIVAS EN COLOMBIA

EMILIO SORZANO ESPINOSA
Calle 69 # 6-02 apto 606
tel 2832683-2362123

En Colombia ha incursionado la Informática Médica en sus diferentes modalidades.En un principio fué estrictamente administrativa,para ir poco a poco desarrollándose en las diferentes áreas cientificas,diferentes especialidades y en sus diferentes formas.La creación de software en el pais ha disminuido el escepticismo dentro del cuerpo médico,además de comprobar las capacidad para su producción con calidad y profesionalismo científico.Los sistemas expertos cumplen un papel importante en el soporte del diagnostico ya sea directamente con programas dirigidos hacia alguna patología especifica, la lectura experta de laboratorio o de imágenes,o el software especializado en calcular los parámetros matemáticos de algunos conocimientos fisiológicos o patológicos.La simulación de comportamientos tipo tanto en la fisiología como en la patología es de gran utilidad en el diagnóstico,control tearapéutico y en la docencia en las facultades de medicina.Algunas otras modalidades de la Informática como la aplicada a genética calculando los riesgos de tener descendecia enferma,riesgos de malformaciones congénitas,cálculo computarizado de la edad gestacional,manejo epidemiológico de la información con el cálculo del riesgo por zonas y por enfermedad,el diseño de prótesis,control de bombas de infusión,simulación de casos clínicos para la docencia,control de la Unidad de Cuidado Intensivo,cateterismo y radioterapia, son algunos de los aspectos, aparte de los administrativos y de los administrativo-científico que estamos desarrollando en la Informática Médica en Colombia,con unas perspectivas amplias,contribuyendo asi,al desarrollo de la medicina y de las ciencias de la salud en el Pais.En el sector administrativo,hemos desarrollado software a nivel privado y hospitalario,siendo el caso de la sistematización de consulta externa del Hospital San Juan de Dios,ejemplo también de la deficiencia en el presupuesto para adquirir los aparatos necesarios para su funcionamiento.El Estado debe contribuir,facilitando la capacitación de personal influyendo asi el cambio de actitud mental y disminuyendo las malsanas dependencias tecnológicas de paises desarrollados.

El computador como el cerebro humano,poseé las áreas de recepción de información,el proceso de esta y una sección para su almacenamiento en la memoria.Está concebido a semejanza de cerebro con la ventaja de que conocemos completamente su funcioamiento sus reglas y su capacidad.El sistema binario que utiliza el computador ,ara el procesamiento lógico y aritmético,equivale a lo que llamamos en la célula la polarización y despolarización,teniendo esta última la ventaja de manejar umbrales de aceptación en lo que concierne a su capacidad de respuesta.La velocidad de esta respuesta es más rápida en el computador pues la velocidad de conducción de las fibras nerviosas en el organismo humano es menor que la de un circuito eléctrico o uno integrado.

Como sabemos,los computadores han incursionado en forma amplia en los procesos administrativos.En las entidades de salud,los utilizamos en la institución como empresa,procesos que son en la actualidad ampliamente conocidos;aparte de ellos.cabe la concepción en la administración hospitalaria desde el punto de vista científico,comprendiendo por ello,el manejo de la información científica,de historias clínicas,estadísticas de los diagnósticos,terapéuticas,tipo de terapéuticas y sus resultados en cada paciente,sistematización de protocolos de investigación como el caso de la quimioterapia,el desarrollo de una base de datos o banco de tumores para poder realizar estudios prospectivos,y otras aplicaciones en todas la especialidades de la medicina y las ciencias de la salud.Esta área del manejo admonistrativo-científico de una institución de salud está muy poco desarrollada en Colombia,redundando ello en la pérdida de tiempo en la recolección de la información,difícil evaluación de los conceptos investigativos,conductas,diagnósticos y medidas en medicina preventiva.La sistematización de estos procesos de manejo y estudio evaluativo de la información,repercutirá en una mejor atención al paciente,disminuyendo la morbilidad y la mortalidad,contribuyendo también a un desarrollo integral y rápido en la medicina investigativa,en la definición de conductas terapéuticas y epidemiológicas.

Dentro de la concepción de la Informática Médica en las diferentes áreas científicas y sus aplicaciones,tenemos que referirnos a la utilización del computador y a la creación de software especializado para esta área,tanto a nivel hospitalario y sus diferentes departamentos médicos,como en la consulta y en la docencia.

Los sistemas expertos,juegan un papel importante en diferentes áreas científicas de la medicina;Como soporte en el diagnóstico,lectura de exámenes especiales,cálculo de parámetros matemáticos (ej.hemodinámicos),procesamiento y lectura de imágenes,son entre otras,algunas de las aplicaciones en diferentes campos,siendo de gran utilidad y validez ,y en algunas ocasiones con la premisa de que estas herramientas sean creadas por grupos interdisciplinarios en que el conocimiento sea válido,depurado y universal en cada tema del que se trate.

En el área específica del diagnóstico,los sistemas expertos deben entenderse como una herramienta válida,pero dependiente del criterio del médico usuario y de cada caso especifico.En las áreas rurales de nuestro pais,pueden jugar un papel importante para los médicos que están aislados del medio científico,si se logra instalar una red computarizada en los hospitales basicos del Sistema Nacional de Salud,red que puede estar dentro del plan Nacional de Información,como lo planteamos hace pocos días en la ciudad de Bogotà.

En la lectura experta de señales biológicas como el electrocardiograma,electroencefalograma,de exámenes especiales como el liquido cefalorraquideo,gases arteriales,espirometria y en la lectura e interpretación de imágenes,los sistemas expertos están jugando un papel importante de ayuda y soporte a las tareas cuotidianas del médico en el proceso diagnóstico.
Estos mismos sistemas son de gran importancia en radioterapia,calculando las dosis de irradición para el órgano especifico a irradiar o sea el órgano blanco y cálculo de dosis de la irradiación que recibirán las zonas sanas.con el mosmo programa y con base en la configuración exacta de area corporal que se va a irradiar,calcula igualmente la sección corporal por donde deben entrar los rayos para lograr asi el menor porcentaje de irradiación del los tejidos sanos y asi disminuir al máximo los efectos secundarios debidos a dicha irradiación.

En anestesia y en las unidades de cuidado intensivo,son de gran utilidad,llevando los récords y curvas de los parámetros que se registran y controlan durante el acto quirúrgico y en las unidades de cuidados intensivos,aportando conceptos diagnósticos y conductas terapéuticas de urgencia.

Las anteriores son algunas de las múltiples aplicaciones en que los sistemas expertos son viables y útiles en el diagnostico médico y en las decisiones de las conductas terapéuticas,facilitando el trabajo lógico y soportando el conocimiento científico del médico usuario,situación que redunda en la mejor atención del paciente y en la disminución de
la morbilidad y mortalidad.

Otro aspecto de la Informática médica nos lleva a la simulación de comportamientos tipo.Esto quiere decir,la simulación de los diferentes funcionamientos fisiológicos (normales) de los diversos sistemas suceptibles de ello en el cuerpo humano.Para su realización,es necesario la creación de modelos matemáticos,
o de otras caracteristicas,modelos no existentes,logrando asi la simulación conveniente para los fines especificos sean ellos,diagnóstico,control terapéutico o docencia.

Como ejemplos,nos referiremos a la simulación del aparato cardiovascular y al pulmonar.

En el primer caso,la simulación de los flujos y las presiones basándose en parámetros como la presión arterial,las inductancias,capacitancias y resistencias,se realiza en cada cavidad cardiaca (aurículas y ventrículos)en el árbol arterial,árbol venoso y árbol pulmonar,con las respectivas aperturas y cierres valvulares.Esta simulación se refiere a la actividad normal del aparato cardiovascular,pero nos permite simular situaciones patológicas de gran utilidad en las unidades de cateterismo cardíaco y realizar un seguimiento diagnóstico y posteriormente terapéutico de las medidas tomadas según la patología de cada paciente.

La simulación pulmonar podemos realizarla en las tres etapas del proceso de respiración como son la ventilación la difusión y el intercambio gaseoso siendo un gran soporte para el diagnóstico de las enfermedades ddependientes de estos tres procesos.

La simulación en general como en estos dos casos específicos,es de gran validez en la docencia.A travez de esta,el estudiante puede captar y entender más fácilmente el funcionamiento de diferentes sistemas orgánicos.

En el caso específico del sistema cardiovascular,la duración del ciclo cardíaco (sístole diástole)es de 0.8 segundos.Podemos simularlo cada 0.04 segundos por ejemplo y así ver la evolución del aumento o disminución de los flujos y de las presiones,su concordancia,y los momentos de apertura y cierre de las válvulas cardíacas,comprendiendo más facilmente la lófica hidráulica en el funcionamiento cardiaco.En estos programas,el estudiante puede crear casos patológicos y comprender los cambios hidráulicos y las consecuencias morfológicas en todo el sistema cardio pulmonar,intimamente ligados en su fisiopatología.

La docencia tiene otras armas en la Informática Médica como es la simulación de casos clínicos en el computador.Para ello,también hemos creado un programa que nos simula ,de un grupo de enfermedades,más o menos 40.000 casos diferentes de una forma totalmente aleatoria,para que no se repitan los problemas presentados al estudiante.El Programa lo califica y al final le compara el caso correcto con el que él desarrolló.De esta manera,el estudiante puede afianzar sus conocimientos en semiología,patología,terapéutica,exámenes paraclínicos, además de profundizar en la logica médica del proceso diagnóstico. El profesor podrá evaluar al estudiante a travez de programa,y cambiar,agregar o borrar enfermedades y sus conductas terapéuticas para así abarcar más áreas dentro de su cátedra.

Aparte de las utilidades a las que nos hemos referido,existen múltiples aplicaciones en otras áreas de la medicina y de las ciencias de la salud.

En genética hemos estudiado y creado software para calcular el riesgo el la descendencia de una pareja de tener alguna enfermedad

genética si hay preexistencia de ella en la familia tanto en las enfermedades recesivas ligadas al sexo,como en las enfermedades genéticas autosómicas dominantes y recesivas.

Dentro del mismo marco de cálculos de riesgo,y de la misma forma,hemos llegado atravez de software especializado a calcular este en las posibles malformaciones congénitas en los neonatos hijos de madres que hayan sufrido durante el embarazo,rubeola,toxoplasmosis,citomegalovirus o hayan sido irradiadas.Esta clase de software especifico tiene gran utilidad en la consejeria genética y prenatal y asi disminuir las enfermedades genéticas y las malformaciones congénitas,contribuyendo además al bienestar de la pareja interesada.

En epidemiología,es invaluable los servicios que presta y puede prestar la informática.En este sector,está encaminada al manejo de toda la información que viene de las seccionales y de las regionales dentro de la pirámide del Sistema Nacional de Salud.Al lograr un manejo óptimo de la información en enfermedades endémicas como la malaria la tuberculosis y la fiebre amarilla entre otras y de las enfermedades inmunoprevisibles,dedicaremos más tiempo al análisis de la información y a las tomas de decisión,trazando asi planes y metas
de educación y vacunación de la población,control de los animales transmisores,siendo posible calcular el riesgo epidemiológico en las diferentes regionales del pais y asi hacer el énfasis necesario en las conductas a seguir.
A travez de estos programas en el sector epidemiológico es posible realizar los mapas de sectores afectados o de riesgo,mapas con las deficiencias en el manejo de las aguas y de las excretas con el consiguiente ahorro de tiempo y poder dedicar este tiempo al análisis,trazar conductas y ejecutarlas,siendo esteel fin lógico de cada regional de salud distribuidas en todo el pais.Incluido dentro de dichos programas,podemos llevar un control del presupuesto y de las donaciones de los diferentes organismos mundiales de la salud,evaluando la concordancia entre el presupuesto asignado y los fines alcanzados para cada programa dentro de la administración de salud a nivel estatal.

El Ministerio de Salud y el Instituto Nacional de Cancerología,comenzarán en breve la creación de un banco de tumores,programa de gran interés a nivel nacional.Con él,se podrán evaluar los diagnósticos y las conductas terapéuticas y hacer un seguimiento de los pacientes afectedos a mediano y largo plazo,teniendo gran beneficio no solo anivel epidemiológico sino a nivel de estudios científicos.

Los computadores además de las aplicacione mensionadas,se están utilizando en diferentes aparatos de ayuda en el diagnostico como

en la Tomografía Axial Computarizada (TAC) los gases arteriales y están en el diseño y control de marcapasos y bombas de infusion;Nosotros en este momento,hemos incursionadfo en el diseño de las prótesis de miembros inferiores con los cálculos específicos de zonas de presión,de apoyo,dolorosas y otras caracteristicas del acople del muñón a la prótesis.

Hemos revisado diferentes aplicaciones de la Informática Médica aparte de los que administrativamente son factibles de desarrollar.Como director del área de Informática Médica de la Universidad Piloto de Colombia y de Centrofuturo (Centro de estudios para el Futuro)y como Colaborador de la Facultad de Ingeniería de Sistemas de la Universidad Nacional de Colombia,he incentivado el desarrollo de la Informática Médica,y con grupos interdisciplinarios hemos realizado diferentes clases de software en las variadas ramas en que hemos subdividido esta aplicación de la Informática siendo algunos de ellos,la simulacion del aparato cardiovascular,pulmonar,renal,de casos clínicos,sistemas expertos en epilepsia,dolor abdominal,calculos de riesgo,manejo epidemiológico de la malaria y de la rabia Sistemas de enseñanza en oftalmología,neumología y medicina interna,manejo experto de las Unidades de Cuidado Intensivo entre otros.Seguimos en la actualidad sobre la misma linea de desarrollo del software especializado tanto a nivel administrativo como científico.

La mayoría de las áreas de esta clase de Informática la estamos desarrollando en Colombia,pero es indispensable que el Estado tome decisiones respecto a la capacitación de personal,incentivar la investigación y adicionar en el pensum de las facultades de medicina la Informática.Esto lograría en un futuro próximo la disminución del excepticismo,tarea titánica de por si,y se lograría un cambio en la actitud mental de los profesionales de la salud.Estas tareas son importantes realizarla pronto,pensando en que en los paises desarrollados fomentan la investigación buscando su futuro,mientras nosotros estamos buscando apenas llegar a nuestro presente.
Vale la pena recalcar que el software creado en colombia es de alta calidad no solo en lo que a sistemas se refiere sino desde el punto de vista investigativo,creando soluciones que nos eviten una malsana dependencia de paises desarrollados,contribuyendo asi a un mejoramiento en la calidad de vida.
El estado no debe ignorar el sector universitario ni el privado,los cuales pueden darle apoyo investigativo y en la fabricación del software sea este para uso privado o estatal.
Los programas educativos y de manejo de la información,el estado debe planearlos a mediano y largo plazo,con continuidad en los diferentes gobiernos,y asi poder recoger sus frutos en eu futuro próximo,igualándonos con paises de alta tecnología y desarrollo.

SISTEMA INTEGRADO DE CONTROL PARA LA UNIDAD DE CUIDADOS INTENSIVOS PEDIATRICA.

EMILIO SORZANO ESPINOSA
Calle 69 No. 6-02 Apto 606
tel 2832683 2362123

CARLOS SANCHEZ FALLA
Clinica Colsubsidio
tel 2126795

MARIA JOSEFA DUQUE B.
Cll 187 No. 29-35
tel 6714470

JULIO ERNESTO CASTRO D.
Cll 12 No. 6A-15 Bosa
tel 7751775

FERNANDO GONZALEZ FRANCO.
Cra 28 No. 4-11 Sur
tel 2375872

Siendo la Unidad de Cuidados Intensivos creada para la vigilancia contínua y el tratamiento inmediato de pacientes gravemente enfermos, cuyas funciones vitales estan gravemente alteradas, poniendo en peligro la vida. El sistema pretende ayudar al médico en el diagnóstico y tratamiento de los pacientes al resolver una serie de problemas que generalmente se presentan en la unidad de cuidado intensivo, como son: La poca legibilidad de los datos del paciente debido a la diversidad de anotaciones por parte del personal (Médicos y enfermeras, en sus respectivas hojas de anotaciones a un paciente), y el acceso rápido a la información. También un archivo de causas y tratamientos a seguir para determinadas patologías que son poco frecuentes, como una serie de calculadores agrupados en los diferentes subsistemas que pueden estar afectados; son de gran ayuda para el medico.
Además permite mediante cálculos estadísticos conocer las tendencias de los principales parámetros, que pueden ser utilizados para optimizar el manejo de los recursos de la Unidad con el fin de dar mejor atención a los pacientes.
Hay que tener siempre presente el cuidado especial que se debe tener en la unidad de cuidados intensivos pediátrica, ya que los niños no son pequeños adultos, sino que son niños y deben ser cuidados como tales.

En la actualidad la Unidades de Cuidado Intensivo pediátricas de la mayoría de clinicas y hospitales de Bogotá, no estan sistematizadas, observando este fenómeno hemos querido contribuir poniendo al servicio de éstas las nuevas tecnologias para facilitar a los medicos y enfermeras el acceso a la información.
Tambien mediante el sistema se pretende en forma generica dar solución a los principales problemas que se pueden presentar en una unidad de cuidados intensivos pediatrica, asi como prestar al personal de la unidad una ayuda practica en cuanto a la consulta del material directamente relacionado con los datos de los pacientes.
Actualmente pueden observarse una serie de dificultades o fallas como es la no existencia de un documento o esccito que recopile las principales, al igual que las menos frecuentes patologias que se pueden presentar en la unidad, con el fin de guiar al medico que en un momento dado no recuer de cuales son los valores normales de dicha patologia o el tratamiento a seguir. Tambien se nota una carencia de ayudas visuales que mejorarian la interpretación de los signos vitales (Tension arterial, Temperatura, Pulso, etc), datos que son llevados principalmente en forma numerica desconociendo las ventajas de un sistema grafico relacional.
Tambien se pretende que los datos historicos sean utilizados para hacer proyecciones que permitan optimizar la utilización de los recursos.

Hemos dividido el sistema integrado de control en Modulos para facilitar su realización y manejo, estos son:
- Modulo de Signos vitales,
- Modulo de Evolución y ordenes medicas,
- Modulo de Laboratorios,
- Modulo de Historias clinicas,
- Modulo de Evolución enfermeria,
- Modulo de recomendaciones terapeuticas, y
- Modulo de Ayudas tecnicas.
- Modulo de Estadisticas

Modulo de Signos Vitales: Modulo mediante el cual se controlan y manipu lan las funciones vitales. Aqui son tomados datos de signos vitales (Tension arterial, temperatura, pulso, pvc,respiración,etc.), neurologia (escala de coma (ojos, verbal, motora), pupilas (diametro, respuesta), y tono muscular), control de liquidos de entrada y de salida.
Todos estos, datos que son tomados regularmente al paciente.

Modulo de Evolución y Ordenes medicas: En este modulo el medico anota tanto la evolución del paciente como el momento en el cual sucedieron cada uno de los cambios, todo esto queda archivado y podra ser consultado luego por los demas medicos de la unidad. Tambien aqui podra anotar el tratamiento a seguir y las ordenes hacia la enfermera indicandole las drogas que se deben aplicar con sus respectivas cantidades, unidad y via, ademas de los examenes que se le deben realizar al paciente en las proxi mas horas. En general las acciones a seguir en el tratamiento para cada uno de los pacientes que se encuentren en la unidad.

Modulo de Laboratorios: Aqui se tienen almacenados todos los datos resul tantes de los principales laboratorios realizados al paciente, asi como los resultados de los analisis separadamente en cada uno de los ventila dores utilizados (ventilador de presion, tiempo y volumen).

Modulo de Historias Clinicas: En este modulo se registran todos los datos de la historia clinica del paciente, tanto si el paciente es enviado por otra area de la clinica u hospital, como si es remitido de otra entidad. Los datos que aqui se registran son: Nombre, Fecha de Nacimiento Residencia, Sexo, Edad, Grupo sanguineo, Rh, Alergias, Enfermedades, etc. Adicional a estos datos basicos el medico hace anotaciones del motivo de consulta, antecedentes, examen fisico (talla, peso, superficie corporal, etc.) e impresiones diagnosticas. Todos estos datos por ser muy importantes y de constante consulta por parte del medico se agruparan con las evoluciones.

Modulo de Evolución enfermeria: en este modulo la enfermera indica que drogas se le aplicaron al paciente asi como las dosis, todas estas que anteriormente fueron indicadas por el medico en el modulo de evolución y ordenes medicas. Esta evolución de enfermeria se hace con el fin de que el medico sepa en todo momento los examenes y drogas que se le han aplicado al paciente.

Modulo de recomendaciones terapeuticas: este modulo se encuentra dividido de acuerdo al problema que se presente, ya sean fisiopatologicos, por falla de sistemas o por problemas especificos. En esencia es una recopilación de las principales patologias que se presentan en la unidad de cuidados intensivos pediatrica y se da al medico en forma resumida las causas y algunas recomendaciones del tratamiento a seguir.

Modulo de Ayudas tecnicas: Este modulo se divide en tres Submodulos, que son: el submodulo de medicamentos, el submodulo de constantes y parametros y el submodulo de calculadores.

En el submodulo de medicamentos se recopilan los medicamentos de mayor uso en cuidados intensivos, los cuales podran ser consultados por el medico en cualquier momento.
Para cada medicamento se tiene información de su nombre generico y presentación, dosis y vias de administración, e indicaciones.
Este archivo de medicamentos puede ser modificado o pueden ser adicionados nuevos medicamentos de acuerdo al deseo del medico o de la institución. Inicialmente cuenta con unas 65 drogas que son las mas usadas para el tratamiento de las patologias mas frecuentes.

En el submodulo de Constantes y parametros se recopilan constantes y parametros utiles en cuidados intensivos, los cuales como en el caso anterior podran ser consultados por el medico en cualquier momento.
Para cada constante y parametro se tiene información del nombre del estudio, producto biologico necesario y de su interpretación.
El medico tambien podra modificar o adicionar a este archivo los datos de las constantes y parametros que el considere necesarios.

El submodulo de calculadores se encuentra a su vez dividido de acuerdo al sistema afectado. Estos son: Cardiovascular, Metabolico, Renal y Nutricional.

En cada uno de estas subdivisiones se tienen un conjunto de formulas que permiten al medico obtener una respuesta rapida de calculos que de otra manera serian muy demorados y menos confiables. A continuación inidicamos

los calculos que el medico podra realizar. Estos son:

- Diferencia a-v 02
- Gasto cardiaco (aproximado)
- Indice Cardiaco
- Shunt clinico
- Shunt estimado (asuma dif. a-v02)
- Resistencia sistemica
- Resistencia Pulmonar
- Indice extracción oxigeno
- Volumen sanguineo circulante
- Cantidad sangre (para elevar HB)
- Calculo de perdida sanguinea
- Osmolaridad calculada
- Anion Gap
- Corrección del sodio
- Depuración de creatinina
- F E N a
- U / P creatinina
- U / P sodio
- U / P osmolar
- Depuración Osmolar
- Depuración de agua libre

- Requerimiento diario de liquidos, calorias, proteinas y grasas.

Dentro del submodulo de calculadores tambien el medico podra realizar un analisis de gases sanguineos.
El cual preguntando: el porcentaje inspirado de oxigeno, tipo de oxigeno, hematocritos, ph, pco2, y po2. Podra de acuerdo a la fuente (Arterial, Capilar, Venosa o No conocida), darle al medico información de los datos de : la saturación de o2, gradiente alveolo arterial, base exceso y otros com sus valores normales y el valor observado.

Luego si el medico lo desea hara una interpretación del po2, indicando si esta sobre los limites normales, en el rango normal, o si esta decre mentada. Luego hara una interpretación del ph indicando si esta dentro de los limites normales o si por el contrario muestra una acidosis o una alcalosis, ademas de indicar en que grado (Marcada, moderada o leve).

Adicional a estas interpretación tomando el valor calculado de hco3, indi cara como se encuentran los niveles de bicarbonato no respirado.

luego, se realizara la interpretación del pco2, indicando si esta normal, decrementado o incrementado y debido a que, ya sea por la hiperventila ción alveolar o la hipoventilación alveolar.

Luego, hara una interpretación del gradiente alveolo arterial, en la cual indica siempre y cuando este pueda ser calculado, su estado ya sea normal o la indicación de un shunt leve, moderado o marcado.

Por ultimo ejecutara una interpretación del ph, indicandole al medico la causa de este ph, ya sea un componente respiratorio normal, una acidosis

respiratoria, una alcalosis respiratoria, un componente metabolico normal, una alcalosis metabolica, o una acidosis metabolica. y el estado en que estas se presentan, ya sea (marcada, moderada o leve).

Modulo de estadisticas o de Estudio de relaciones:
En este modulo se sacan estadisticas de acuerdo a los datos presentados por los pacientes, para el calculo de estas se toman los siguientes datos:

- Sexo.
- Edad.
- Promedio de estancia.
- Procedencia.
  que puede ser: Urgencias, Consulta externa o piso.
- Causa de entrada
  que puede ser: traumatico - infeccioso
  traumatico - no infeccioso
  no traumatico - infeccioso
  no traumatico - no infeccioso
- Sistema afectado
  que puede ser: Neurologia
  Cardiovascular
  Respiratorio
  Gastrointestinal
  Renal
  Metabolico
  Otros
- Causas patologicas
  que puede ser: Medico
  Quirurgico
  Medico - Quirurgico
- Estado final del paciente
  que puede ser: Murio
  No murio

los datos de causa de entrada, sistema afectado y patologias son combinados para tener una mejor vision de las tendencias.

Adicional a estos modulos basicos existen otros dos modulos que son los de :

- Modulo de auditoria
- Modulo de ayudas (manejo).

Que pretenden controlar y ayudar en el manejo del sistema, para que pueda ser manejado tanto por el personal de la unidad que llegara a familiarizarse con el sistema, como los medicos que lleguen a la unidad a realizar observaciones de algun paciente (como es el caso de los especialistas).

Modulo de Auditoria:
En este modulo se registran las personas que pueden accesar las diferentes partes del sistema. Esto con el fin de asignarle a cada uno una restricción o no para realizar determinados procesos y asi proteger la informa

ción, ya que podrian presentarse casos como que un medico por equivoca ción escriba sobre la evolución de otro medico para un paciente, o que una enfermera dañe el archivo de algun medicamento o lo altere, y asi como estas otras tantas circunstancias que pondrian en peligro el buen estado de los pacientes de la unidad.

Tambien se registran para cada proceso que se realiza, la hora y la persona que lo realizo, asi cualquier equivocación puede ser corregida facilmente, por ejemplo adicionarle datos a un paciente a una hora que no correspondia, etc.

Modulo de Ayuda:
En este modulo se tienen archivados para cada proceso una pequeña expli cación, de tal manera que si el medico o la enfermera no recuerda como dirigirse o realizar un proceso especifico, esta es una rapida herramien ta de consulta.

Tambien aqui se encuentran almacenados los valores limites para cada uno de los datos que se le toman al paciente, asi garantizando que en todo momento el me ico o la enfermera podran realizar el proceso deseado sin necesidad de tener un conocimiento muy profundo en el campo de la informatica.

SISTEMA AUTOMATICO DE EVALUACION DIAGNOSTICA EN EDUCACION MEDICA

EMILIO SORZANO ESPINOSA
Calle 69 # 602 apto 606
tel.2832683 2362123
Bogotá.

ALBERTO BERNAL
CARLOS RODRIGUEZ
Universidad Piloto
Bogotá.

Este programa está enfocado a la docencia en las facultades de medicina sin tener en cuenta la especialidad.El sistema realiza la simulación totalmente aleatoria de las enfermedades implementadas en su base de datos relacional,permitiendo al estudiante ejercitar sus conocimientos en semiologia clinica,patologia,laboratorio y terapéutica,además de afianzar sus conceptos en la lógica médica del proceso diagnóstico.
El sistema simula los diferentes pasos de la enfermedad,pero es necesario que el estudiante vaya solicitando la información en el orden correcto y llegar al diagnóstico y la conducta terapéutica adecuada.Al final,el sistema emite el diagnóstico correcto y una nota sobre 100 del caso realizado por el estudiante,comparándole además el caso creado por el computador y el desarrollado por el estudiante.Un módulo aparte es el del profesor o administrador del sistema,con el cual puede adicionar enfermedades,borrar las existentes,o cambiar cualquier información referente a ellas y a los estudiantes que tienen acceso al programa.Dada esta característica,se puede utilizar en cualquier especialidad médica o paramédica y mantiene su vigencia en el marco docente,con datos actuales o enfermedades recientemente descritas.

La informática aplicada a la docencia en las facultades de medicina es una herramienta que permite mayores proporciones aunque en nuestro medio,hasta ahora,ha sido poco utilizada.

En la enseñanza médica son varias las modalidades de software que se pueden utilizar;en este caso nos referimos,hemos creado un programa de simulación de casos clínicos en el computador relacionados directamente con el área de medicina interna.Con él,el estudiante tiene la posibilidad de autoevaluarse,repasar sus conocimientos en semiología,fisiopatología,además que le permite establecer las conductas diagnósticas y terapéutica a seguir.
El programa ,en un principio,se ha implementado con 30 enfermedades,las que puede simular aleatoriamente hasta constituir 40.000 casos diferentes.De acuerdo con esto es difícil que el estudiante encuentre el mismo caso en dos oporunidades.

Teniendo en cuen cuenta que el programa busca el desarrollo de la lógica médica,el estudiante debe recorrer las diferentes etapas del proceso diagnóstico en su orden;Anamnesis:signos,sintomas,antecedentes y revisión por sistemas.Posteriormente pasa al examen físico seguido del diagnóstico diferencial.Posteriormente sigue la solicitud de exámenes paraclínicos,sean de laboratorio o procedimentales para llegar a un diagnóstico final.Enseguida,el estudiante debe decidir una conducta terapéutica siendo esto el final de la simulación del caso.

El sistema sigue la misma linea en el orden correcto y va evaluando las decisiones que el usuario va tomando paso a paso,para asi emitir una nota final del proceso simulado.

El programa le facilita al estudiante los sintomas y signos correspondientes a cada sistema orgánico pero solamente por el que el usuario pregunte.De la misma manera,presenta en pantalla los resultados de los exámenes que el estudiante haya solicitado,dejándole a este su interpretación.
Finalmente,cuando el estudiante toma una decisión diagnóstica,se le permite revisar de nuevo todo el caso y realizar las correcciones que crea necesarias.Por último,el programa emite una nota ya sea por pantalla o por la impresora,le dice el diagnóstico correcto y le hace una comparación del caso creado por el computador y el desarrollado por el estudiante,asi este puede evaluar sus fallas y corregirlas en un futuro.

El sistema poseé una base de datos relacional con toda la información de la enfermedad,síntomas,signos,antecedentes,exámen físico,laboratorio,exámenes especiales ,diagnóstico y terapéutica.Las enfermedades implementadas son:

Aneurisma de la aorta abdominal	Apendicitis
Aneurisma aorta torácica	Bronquitis aguda
Bronquitis crónica	Cáncer de estómago

Carcinoma Broncógeno
Comunicación interauricular osteum secundum
Cor pulmonale crónico
Encefalopatía Hepática
Enfisema Pulmonar
Fiebre reumática aguda
Hemofilia
Hiperplasia prostática
Insuficiencia cardíaca aguda
Isquemia miocárdica prolongada con signos objetivos de infarto
Isquemia miocárdica prolongada sin signos objetivos de infarto
Leucemia aguda
Miocarditis
Pericarditis aguda
Pielonefritis aguda
Tuberculosis pulmonar
Ulcera gástrica.
Carcinoma de próstata
Diverticulitis
Enfermedad de Hodgkin
Embolía pulmonar
Glomerulonefritis aguda
Hepatitis aguda viral
Infección de las vías urinarias
Miocardiopatía restrictiva
Nefropatía obstructiva
Pericarditis constrictiva
taquicardia ventricular
Ulcera duodenal

Al final del proceso,en el momento de conocerse el diagnóstico correcto,el programa presenta en pantalla la definición completa de la enfermedad que se simuló.

La escogencia de síntomas y signos se basa en una división de síntomas principales y secundarios para cada enfermedad,siendo igual el método para los signos.De esta manera,es posible lograr una gran combinación de estos elementos y asi crear casos diferentes de la misma enfermedad como ocurre en la realidad.Para los exámenes de laboratorio,los datos normales o patológicos,según el caso,son escogidos por el sistema entre una gama correspondiente a los rangos ya sean de normalidad o de anormalidad.Estas caracteristicas simulan también lo que sucede en la realidad,permitiendo que la simulación efectuada en el computador sea lo más cercano al desarrollo de una consulta médica.

Teniendo como soporte la base de datos relacional,el sistema selecciona aleatoriamente la enfermedad.Este módulo a su vez,encadena el de sintomas y signos,luego el de los exámenes de laboratorio y de procedimiento y finalmente el que contiene las conductas terapéuticas,comenzando en este momento la ejecución del caso simulado.

El sistema se presenta en pantalla en forma de menús.El primero de ellos permite la ejecución del caso,el segundo facilita el acceso a las ayudas y a las guias de funcionamiento del sistema.
En cada proceso que se lleve a cabo en cualquier etapa de la

simulación,se presentan menús diferentes y la ayudas necesarias para el buen funcionamiento del sistema.

El programa además de los módulos descritos,tiene uno especial para el administrador del sistema o sea el profesor.Con él,el docente

puede agregar enfermedades,modificar la información almacenada o borrarla completamente y cargar nuevos problemas clínicos,dando la posibilidad de ser utilizado en cualquier especialidad médica con patología variada y cambiante.
Además,el profesor podrá grabar los códigos de los estudiantes que pueden tener acceso al programa sin que ellos puedan alterar cualquier información.Este sistema,vale la pena aclarar,por la forma como está concebido,sirve como lo hemos dicho,para la docencia en las facultades de medicina pero,también en las escuelas y facultades de ciencias paramédicas.

Esta característica,hace que el software que hemos creado no pierda su validez,mantenga su vigencia y los casos sean ilimitados.
Con este programa,hemos querido contribuir al desarrollo de la informática médica,específicamente en lo que se refiere a la docencia en las facultades de medicina.Esta es una forma,además,de crear la inquietud en las escuelas médicas de abrir en el pensum académico la cátedra de informática medica.

ANALISIS MATEMATICO Y CALCULO DE RIESGO EN ENFERMEDADES GENETICAS

EMILIO SORZANO ESPINOSA
Calle 69 # 602 Apto 606
tel 2832683 2362123

ALBERTO MARTINEZ
Centro de cómputo
de la Universidad Piloto

MARCELA TRIANA

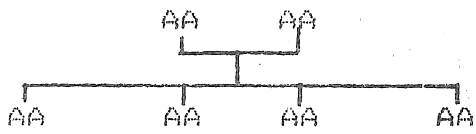
Asesor engenética

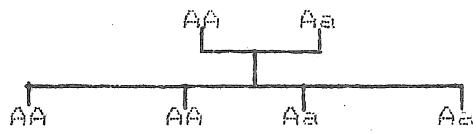
JAIME BERNAL V
Hospital de san Ignacio
Sección de Genética

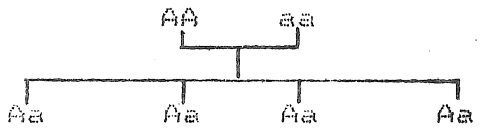
Este trabajo comprende programas que se refieren a la aplicación de los sistemas en la medicina,específicamente en el área de la genética.Están encaminados a brindar asesoria tanto al médico como al paciente en la consejeria genética.Estos permiten la creación de las historias genéticas de una forma gráfica con la simbología universal,archivar las historias,y realizar los cálculos apriori,aposteriori,la probabilidad conjunta,condicional y finalmente el riesgo de ocurrencia de la enfermedad genética que se esté estudiando.Permite además,tener los reportes por pantalla e impresora y archivar dicha información para estudios científicos posteriores.Los cálculos se realizan en las enfermedades recesivas ligadas al sexo y en las enfermedades autosómicas dominantes y recesivas.
Poseen además,diferentes modulos para la creación de la historia,su modificación,su archivo,su recuperación o eliminación, las ayudas para su funcionamiento y un banco de enfermedades genéticas.
Dedemos subrayar,que además de explicar su funcionamiento,realizaremos la demostración del programa con la creación de casos entre los asistentes,ya que el sistema está totalmente concluido y validado.

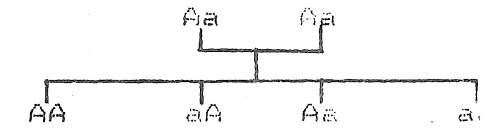
En la consulta genética es de gran importancia la consejería que se le puede dar a una pareja con problemas genéticos dentro de la familia de alguno delos dos o de los dos conyuges.Esta se basa en el estudio de la enfermedad,su forma de herencia y la presencia de dicha enfermedad en los diferentes componentes del árbol genealógico de la persona o personas que consultan.Pensando en los diferentes procesos,hemos creado un programa que comprenda la consulta genética y los cálculos del riesgo para la descendencia de una pareja.

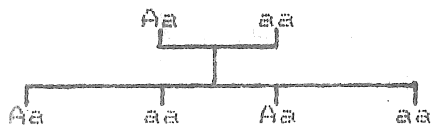
Hemos dividido en dos programas diferentes los cálculos del riesgo para las diferentes clases de herencia.
El primero está encaminado al cálculo del riesgo y el archivo de la historia genética en las enfermedades recesivas ligadas al sexo.El segundo está dirigido a las enfermedades autosómicas dominantes y recesivas.Para entender estos conceptos,pasemos a su explicación.
El ser humano hereda 23 cromosomas del padre (espermatozoide) y 23 cromosomas de la madre (óvulo).Así se crean 23 pares de cromosomas o sea 46 en total en cada célula que se va formando luego de la fecundación del óvulo por el espermatozoide.
Admitamos primero,que el gene A es dominante sobre el gene a,y que el gene A es patológico y porta una enfermedad.
Otro concepto es el de Homocigoto y heterocigoto. Homocigoto es cuando existe el mismo alelo en un locus determinado en cromosomas homólogos.Heterocigoto es cuando tiene alelos diferentes en un locus determinado en cromosomas homólogos.
consideramos homocigoto es que en la misma pareja de cromosomas y en cada uno de ellos,haya un alelo patológico idéntico.Lo heterozigoto,es cuando un alelo patológico está en diferente locus con respecto al otro cromosoma que forma la pareja..Definimos así estos conceptos con el ánimo de hacernos entender por los profesionales no médicos.
En el caso de la dominancia,de una persona o personas con alelos dominantes,en la población humana podemos tener las siguientes combinaciones:

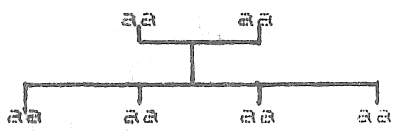






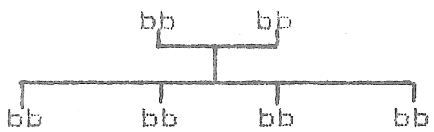


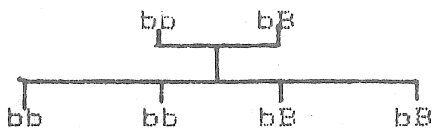


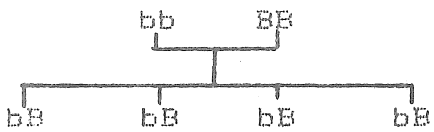


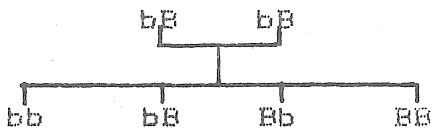
En todos estos casos posibles de descendencia,los hijos portadores del alelo "A" serán enfermos de la enfermedad de que se trate.

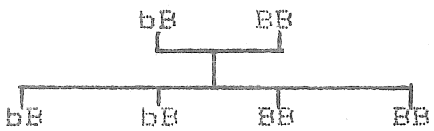
En el caso de la recesividad consideremos el alelo "b" patológico encontrando las siguientes posibilidades:

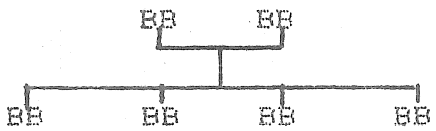






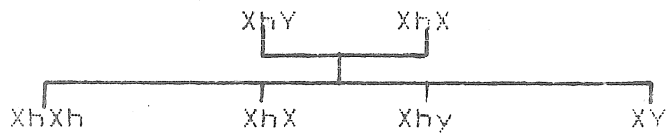






En estos seis casos tendremos que los enfermos en la descendencia solo serán los que posean los dos alelos "bb".los que tienen solo un alelo patológico,los consideramos portadores pero no sufren la enfermedad.

En nuestro primer programa en las enfermedades recesivas ligadas al sexo,quiere decir que la enfermedad se transmite en el cromosoma "X"del sexo.Siendo las mujeres XX y los hombres XY comprendemos que si es una forma de herencia recesiva,las mujeres serán portadoras,siendo X el alelo enfermo en un solo cromosoma de la pareja que determina el sexo.En los hombres que lleven el alelo enfermo X y no el sano x,serán enfermos.Es el caso de la hemofilia.Llamemos el alelo enfermo de la hemofilia por la letra h.La unión entre un hombre hemofílico y una mujer heterocigota clínicamente sana nos llevará al siguiente esquema:



En este caso los unicos sanos son la mujer XhX y elhombre XY.Sinembargo,la mujer anterior es portadora.Los otros dos,hombre y mujer son enfermos.

Nuestro segundo programa se refiere a las enfermedades autosómicas,que se transmiten en los demás cromosomas que no son los de sexo,ya sean dominantes y recesivas.En este programa,el sistema reconoce el tipo de herencia (dominante o recesiva).Si no es claro el patrón en el árbol genealógico creado emite el mensaje correspondiente para que el usuario esté prevenido y no incurra en errores.

Los dos programas poseen diferentes módulos para cada función especifica.El módulo de creación del árbol genealógico presenta las posibilidades de hacerlo en la pantalla de una forma gráfica con la simbología universal en genética.Este módulo permite crear la historia genética con su pedigree;dicha información es susceptible de modificarse parcial o totalmente y de almacenarla en la memoria del programa.
En el caso específico del módulo de cálculos permite realizarlos con diferentes métodos;teniendo información positiva (patológica)anterior (al consultante,información positiva posterior o información positiva anterior y posterior.
Al respecto de las enfermedades autosómicas dominantes y recesivas se realizan los cálculos en los genotipos desconocidos.
Los cálculos tienen como base el teorema de probabilidades de Bayes,que calcula las probabilidades apriori aposteriori,condicional,conjunta y finalmente el riesgo de ocurrencia de la enfermedad que se esté investigando.

El siguente modulo es el que maneja los archivos, posibilita el acceso o la recuperación de la información de la historia clínica genética,además de facilitar la consulta de información adicional de esta sección del sistema.

El cuarto módulo permite generar los informes ya sea por pantalla o por impresora para que se archiven con la historia clínica física.

Con este programa específicamente pretendemos contribuir a nivel científico en la consulta médica,ya sea especializada en el caso del genetista o a nivel de medicina general.De esta manera,se pueden dar respuestas rápidas y concretas sobre problemas en la descendencia de los pacientes,asi se disminuye la probabilidad de ocurrencia de enfermedades genéticas.

Este sistema se puede implementar dentro de un paquete de programas dirigido también a los médicos rurales,teniendo en cuenta las limitaciones y escasos recursos con que cuentan los facultativos en estas zonas apartadas del medio científico colombiano.

MODELO PARA EL CONTROL Y VIGILANCIA EPIDEMIOLOGICA DE LA RABIA.

AUTORES:

GERMAN A. PINZON C.	Cra 22 No. 54A-49	Tel: 2173120
MAURICIO LARA R.	Cra. 40 No. 24-64	Tel: 2684182
EMILIO SORZANO E.	Cra. 8a. No. 15-73	Tel: 2122978

Al tomar conciencia de la importancia de las enfermedades virales, que afectan al ser humano directa o indirectamente, el modelo hace un aporte en el área de la medicina, específicamente con la enfermedad de la rabia llevando un control adecuado y una constante vigilancia epidemiológica que permiten tomar acciones y medidas rápidas en contra de esta enfermedad mortal para el hombre.

Para lograr esto, el modelo recibe datos, los procesa y proporciona información veraz y oportuna en el momento en que la Sección de Zoonosis del Ministerio de Salud lo requiera, agilizando los procesos que permitan un efectivo control y una adecuada vigilancia epidemiológica.

Además, el modelo mantiene actualizado el conocimiento del comportamiento y distribución de la enfermedad de la rabia, a partir de los datos recibidos de las diferentes seccionales del país, que son procesados en la Sección de Zoonosis en una forma automatizada generando estadísticas referentes a la situación de la enfermedad, como también calculando índices epidemiológicos como son: tasas, proporciones, y de acuerdo a estos resultados la Sección de Zoonosis toma las medidas de control que se requieran.

Nota: Para poder apreciar los resultados, el software será exhibido el día de la conferencia.

A nivel nacional, a través de los diferentes Servicios Seccionales de Salud se recolecta la información que requiere el Ministerio de Salud, para poder llevar a cabo todas las medidas necesarias para la prevención de brotes epidemiológicos de la rabia. Sin embargo a ésta información se le daba un manejo inadecuado ya que las diferentes variables que recopilaba el sistema no eran consolidadas, tabuladas, ni analizadas en forma integral, debido a su volúmen y a la falta de una herramienta que agilizara el manejo de estos procesos, que en el pasado se llevaban en forma manual.

El tiempo que transcurría desde que la información recopilada entraba a los procesos de consolidación y tabulación hasta la presentación de los resultados del análisis con el fín de tomar las acciones de control era considerable, respecto a la evolución epidemiológica de la enfermedad, es decir, que transcurrido cierto lapso la actividad de control que debería ser implementada en determinada región ya no era efectiva.

En vista de la importancia que tiene la lucha contra la enfermedad de la rabia, para la prevención de la enfermedad humana se hacía necesario un sistema automatizado que agilizara los procesos que se llevaban a cabo para obtener una información adecuada, concisa, actualizada, confiable, sistemática y oportuna, que permitiera un proceso de observación y análisis rutinario de la ocurrencia y distribución de la enfermedad de la rabia, para la toma oportuna de acciones de control.

Por este motivo el modelo para el Control y Vigilancia Epidemiológica se define como un sistema automatizado que implementa los mecanismos necesarios para mejorar:

- el manejo de la información,
- operación y ejecucion de las acciones,
- relación tiempo-evolución de la enfermedad,

siendo estos la base para la investigación, planeación, ejecución y evaluación de las medidas de prevención y control de la enfermedad de la rabia en el hombre y en los animales.

Este modelo se caracteriza por ser un modelo estadístico ya que sustenta sus planteamientos en procedimientos estadísticos tanto descriptivos como inductivos:

- Descriptivo en cuanto incluye la representación de los datos en cuadros, lo mismo que síntesis numéricas de frecuencias, tasas, razones y proporciones aplicadas a:

  * Personas expuestas según especie
  * Animales observados
  * Cerebros animales investigados
  * Casos de rabia humana
  * etc.

- Inductivo en cuanto permite formular generalizaciones respecto

a una poblaciòn, sobre la base de datos numèricos observados en una muestra extraida de la misma. Este proceso se lleva a cabo a partir de los diferentes informes generados por el sistema que son analizados por las personas responsables de tomar las correspondientes medidas de control.

El modelo con base en enfoque epidemiològico mide la cantidad de personas expuestas, casos de rabia humana, en general "casos" y la forma como se distribuyen en una poblaciòn; para poder cumplir con ello requiere el apoyo fundamental de la estadìstica.

Esta informaciòn puede ser actualizada, modificada, consultada y borrada a travès de la pantalla, reseñando tambièn que los informes generados periòdicamente son editados por impresora, habiendo informaciòn que puede ser accesada por pantalla en el instante en que se necesite.

El objetivo general del modelo es servir de herramienta fundamental al Ministerio de Salud, en la Seccion de Zoonosis, entregàndole informaciòn que le permita llevar a cabo un eficiente control y adecuada vigilancia epidemiològica de la rabia, teniendo como propòsito final servir a la comunidad.

Asì mismo podemos destacar los siguientes objetivos especìficos:

- Recibir, procesar y proporcionar informaciòn veraz y oportuna en el momento en que se necesite y sea adecuada para la toma de decisiones.

- Agilizar los procesos que permitan un efectivo control y una adecuada vigilancia.

- Mantener actualizado el conocimiento del comportamiento y distribuciòn de la enfermedad de la rabia en las diferentes seccionales.

- Conocer los riesgos necesarios para efectos de los anàlisis epidemiològicos.

- Generar informaciòn que permita formular medidas adecuadas de control para ser aplicadas en las diferentes seccionales.

- Generar estadìsticas referentes a la situaciòn de la enfermedad en las diferentes seccionales y asì poder evaluar las medidas ejecutadas.

- Satisfacer los requerimientos del usuario generando informes mensuales, ocasionales y anuales.

Debido al volùmen de datos que genera el Programa Antirrabico, para efectos de llevar el Control y Vigilancia Epidemiològica de la enfermedad de la Rabia, se hace necesario organizarlos en forma òptima de tal manera que un mismo dato sea aprovechado en las diferentes aplicaciones que ofrece el modelo, en el instante en que se necesite; por esto se hace necesaria la creaciòn de una

base de datos que permite tener almacenados los datos utilizados por el "Modelo para el Control y Vigilancia Epidemiológica de la Rabia", en un conjunto de archivos interrelacionados, evitando redundancias innecesarias, ofreciendo información oportuna y veraz al usuario que lo requiera y en el momento que lo necesite.

El primer paso en la creación de la base de datos del "Modelo para el Control y Vigilancia Epidemiológica de la Rabia", es identificar los diferentes conjuntos de datos manejados por el Programa Antirrábico para efectos de la vigilancia y el control de la enfermedad, así como también establecer las relaciones lógicas que entre ellos existen.

A continuación se describen los conjuntos de datos o módulos que componen el modelo:

- Seccional: Este conjunto de datos contiene información sobre población animal y población humana de cada una de las seccionales en que está dividido el país.

- Poblacion Afectada: Este conjunto de datos encierra información del Programa Antirrábico adelantado en las diferentes seccionales referente a:

  * personas expuestas
  * tratamientos antirrábicos humanos
  * vacunación y eliminación animal
  * etc.

- Disponibilidad de Vacunas: Este conjunto de datos hace mención a la cantidad de vacunas tanto animal como humana que se encuentran disponibles en las seccionales.

- Casos Humanos: Este conjunto de datos contiene información de los casos individuales de rabia humana y complicación neurológica, incluyendo todos los pormenores tanto de la enfermedad rábica humana como de la complicación neurológica que se presente.

El modelo trabaja con una base de datos de tipo relacional ya que se hace necesario indexar los archivos a través de tablas, requerimiento que permite un alto grado de recuperación de los datos que en ellas se encuentran almacenados; por lo tanto la estructuración de las tablas que son manejadas dentro de este modelo permite satisfacer dichos requerimientos.

Los elementos necesarios para la implementación del Modelo para el Control y Vigilancia Epidemiológica de la Rabia están constituidos por el Hardware y Sofware adecuados para el manejo del mismo.

El hardware está constituido por el equipo así como todos los aditamentos indispensables tales como: dispositivos de entrada/salida; es así como el Ministerio de Salud, en la Dirección de Saneamiento Ambiental cuenta con el siguiente

equipo:

- Microcomputador IBM-XT con 640 Kbytes de memoria principal.

- Dispositivos de Entrada/Salida: un drive para diskettes de 5 1/4 pulgadas, doble cara, doble densidad; un disco duro de 20 Mbytes de capacidad; una impresora EPSON de 160 caracteres por minuto.

El software está constituido por el conjunto de programas que permite el funcionamiento del computador. Las especificaciones del Sistema son:

- Sistema Operacional: DOS (Versión 2.1).

- Paquetes: DBase III, llamado Sistema Manejador de Bases de Datos Relacionales.

El diseño del Sistema permite que su funcionamiento se lleve a cabo con base en el uso de menús, ya que estos proveen al usuario de una serie de opciones que abarcan los elementos contemplados en el diseño del mismo, facilitándole la selección del procedimiento que desea realizar.

Para finalizar la presentación del modelo podemos concluir que:

- El modelo permite agilizar el manejo de la información constituyéndose como herramienta fundamental para el logro de los objetivos del Programa Antirrábico, entregando información oportuna, veraz, concisa, adecuada, en el momento en que el usuario del sistema lo requiera.

- Así mismo el modelo entrega una serie de índices epidemiológicos (tasas, riesgos, coberturas) con el propósito de proporcionarle al usuario informes de la situación de la enfermedad de la rabia para cada seccional, para un período determinado, manteniendo actualizado el comportamiento de la enfermedad de la rabia.

- Con los resultados que el modelo entrega a la Sección de Zoonosis, ésta puede formular las medidas adecuadas de control que deban ser tomadas en los diferentes niveles, y además, evaluar las medidas de control ejecutadas con anterioridad.

- El modelo sirve como marco de referencia a futuros proyectos que se desarrollen en relación a otras zoonosis.

SISTEMA DE CONTROL DIGITAL PARA EL MUSCULO ESQUELETICO

AUTORES

JORGE ORLANDO CASTRO NOVA
MANUEL VICENTE PARRADO LOZANO

DIRECCION

Cll 37s * 57-53

0. INTRODUCCION

Algunas veces puede haber en el individuo pérdida de la movilidad de alguno de sus miembros, no obstante conservar las características fisiológicas intactas, al no existir un control motor.

La presente investigación involucra un desarrollo teórico-experimental que consta de varias etapas. La parte experimental comprende tanto la determinación de las características de estimulación para el desarrollo de fuerza en el músculo gastrocnemio de sapos "bufos", como la obtención de un modelo matemático representativo del comportamiento del músculo basado en el análisis de la respuesta del mismo a entradas senosoidales en diferentes frecuencias.

El modelo matemático obtenido caracteriza el comportamiento dinámico del músculo y permite la aplicación de técnicas de control. Este modelo se representa por medio de una función de transferencia la cual fué sometida a pruebas de simulación en el computador, contrastando su comportamiento con el mostrado por el sistema real, para verificar su validez.

Con el modelo del músculo se plantea y diseña un sistema de control digital para regular el nivel de fuerza desarrollado. El esquema obtenido se simula en el computador para obtener sus características de operación y verificar que cumple con las especificaciones de respuesta propuestas.

Se pretende que el trabajo realizado en esta investigación sirva de base en el desarrollo de diversos sistemas que pretendan compensar la pérdida del control motor de los músculos periféricos en los humanos. Aunque se hace necesario la realización de estudios adicionales, esta forma de ejercer un control sobre la actividad muscular es altamente promisoria tanto en los procesos de rehabilitación, como en tratamientos de los diversos tipos de parálisis en los cuales son necesarios un cierto grado de exactitud y estabilidad.

1. FUNDAMENTOS DE ACTIVACION MUSCULAR

1.1 MECANISMOS DE CONTROL

El movimiento es un proceso que involucra multiples niveles motor-sensoriales dispuestos jerarquicamente de acuerdo con su función (Figura 1).

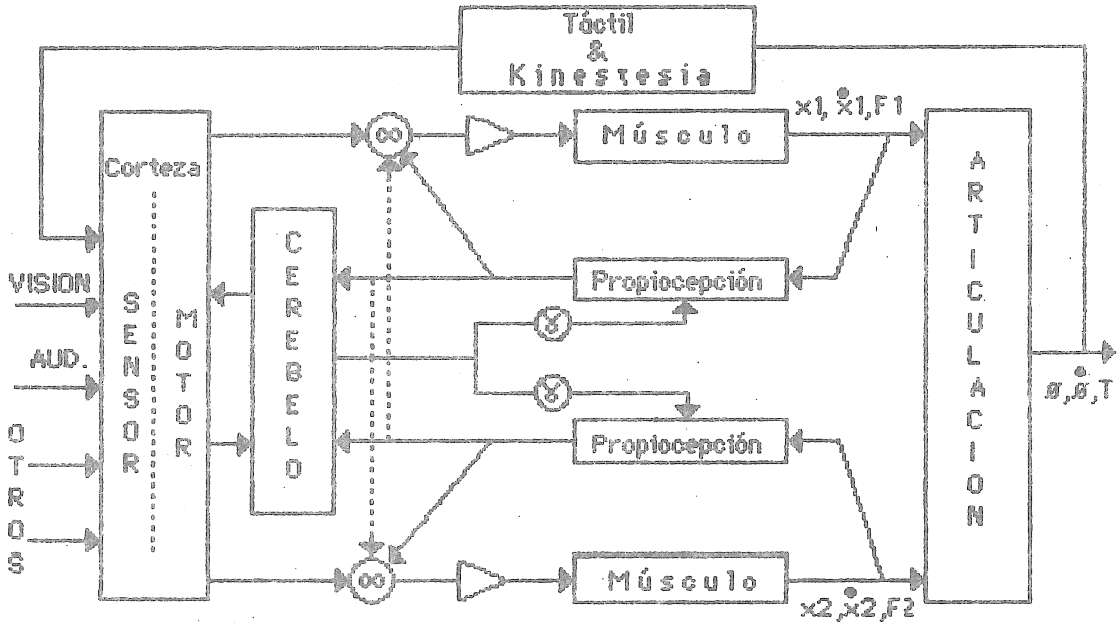


FIGURA 1 Representación esquematica simplificada de los componentes motor-sensoriales del Sistema del Movimiento.

Anatomicamente, es en la corteza cerebral donde la integración de diversas entradas sensoriales conforma la base de la decisión para disparar un movimiento. La señal desciende a través de la médula espinal hasta las motoneuronas alfa y los músculos asociados, mientras simultáneamenta, se activan diversos ciclos sensoriales en el cerebelo, tallo cerebral y médula espinal.

En el ciclo de más bajo nivel, que se compone de una unidad motora y receptores propioceptivos[1], el huso muscular y el órgano tendinoso de Golgi realimentan respectivamente con la información de la longitud y fuerza del músculo a las motoneuronas alfa (2).

Un ciclo de más alto nivel, consta de la proyección ascendente de la misma información propioceptiva de longitud y fuerza al cerebelo. Este ciclo reflejo es capaz de iniciar o modificar el movimiento sin interferencia voluntaria directa desde la corteza motora (4).

El nivel más alto esta constituido por la información táctil y Kinestésica[2] que va a la corteza sensorial via tallo cerebral y tálamo para la percepción, integración y posible modificación voluntaria del estado actual de un miembro.

---

[1] Terminales nerviosas sensoriales que sumunistran información concerniente a los movimientos y posición del cuerpo.

[2] Da la noción del movimiento ejecutado, del esfuerzo, de la situación ocupada á cada instante por los miembros.

Fisiológicamente se distinguen cinco procesos interactivos, activados al iniciarse y durante el movimiento, que se desarrollan en dos niveles; en la articulación y en el músculo.

A nivel de la articulación, el movimiento caracterizado por el desplazamiento angular y el torque (ambos regulados bajo condiciones de carga variables) es controlado por :

- La Sinergía[3] del par muscular agonista-antagonista .
- La Realimentación táctil y kinestésica.

A nivel del músculo, las características de longitud y fuerza son gobernadas por:

- La tasa excitatoria del potencial de acción, APFR[4] (Action Potencial Firing Rate), a nivel de unidad motora.
- El reclutamiento[3] de unidades motoras.
- La realimentación propioceptiva.

De los procesos de movimiento anteriormente mencionados, excluyendo la realimentación propioceptiva, el APFR y el reclutamiento son los dos únicos activos en el control y graduación de la fuerza muscular. Diferentes autores (2), (4), (20), (24) han demostrado la efectividad del proceso de Reclutamiento en el ajuste de nivel de fuerza muscular, tanto así, que se considera funcionalmente necesario para la ejecución de un movimiento.

1.2 ESTIMULACION DE ALTA FRECUENCIA

En un músculo bajo condiciones fisiológicas normales, el control se ejerse mediante la graduación uniforme de la fuerza contráctil. La explicación generalmente aceptada es que una tasa variable de pulsos en los axónes nerviosos individuales, ocurre en combinación con el reclutamiento de los axones multiples que inervan los grupos musculares rápidos y/o lentos (4).

En un músculo intacto pero desconectado del sistema central, el control motor puede ser restaurado parcialmente mediante la estimulación externa del nervio. Las técnicas de mayor difusión que utilizan señales eléctricas son:

- Estimulación mediante un electrodo (17). La señal proporcionada lleva a un nivel de tetanización al músculo, el cual muestra un perfil de fuerza contráctil uniforme y lineal. La frecuencia crítica de la señal esta en el rango de 40-70 Hertz (baja frecuencia); y nos da un control aceptable entre el 90-100% de la fuerza máxima disponible. Niveles de fuerza más bajos no son posibles; por tanto, se tiene un control ON-OFF.
- Estimulación mediante un conjunto de electrodos (22): Los electrodos son colocados en un grupo muscular de tal manera que al ser estimulados secuencialmente se generan diferentes niveles de fuerza, lo cual es análogo a la producción de fuerza digital o discreta. Esta técnica nos provee de un control uniforme y continuo pero aún tosco e insuficiente. Fisiológicamente es probable un daño en el tejido muscular.

---

[3] Acción combinada y simultanea de dos o más factores (músculo), para realizar un trabajo común.

[4] Proceso que inicialmente fija el límite de fuerza máxima que se dispone de un músculo, y tiene efecto minúsculo en la graduación de la fuerza a través de dicho rango.

[5] Activación secuencial de unidades motoras de acuerdo con su tamaño, iniciando con las más pequeñas y sumando sucesivamente las más grandes.

Un principio diferente para tener un control gradual y continuo de la fuerza es introducido por la llamada **Estimulación de alta frecuencia**. El principio en mención induce primero un fondo de fuerza, que se atenua mediante un proceso directo sobre el nervio, proporcionando una replica artificial del reclutamiento concomitante con el APFR; para así obtener una perfecta resolución de fuerza contráctil. Este tipo de estimulación se ajusta más al proceso fisiológico en condiciones normales (2), pues como se menciono anteriormente , el reclutamiento y el APFR son los dos únicos procesos que gobiernan la activación muscular.

El reclutamiento se logra mediante la variación en amplitud de la señal bloqueadora (Figura 2); a mayor amplitud menor sera la fuerza resultante y viceversa. Esta forma de estimular el músculo permite que las unidades motoras pequeñas hagan su mayor contribución en una contracción débil, como sucede bajo condiciones fisiológicas normales.

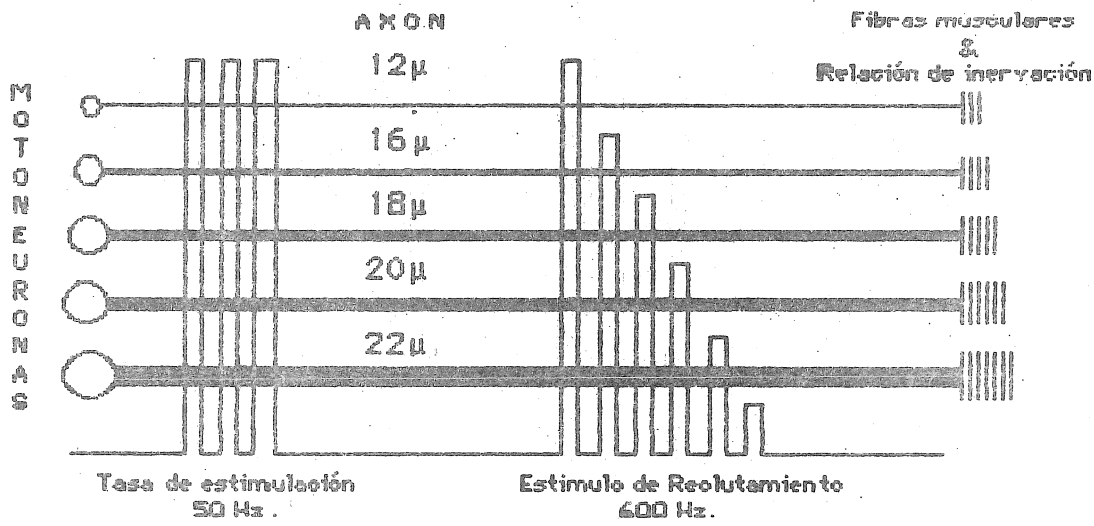


FIGURA 2 Representación esquematica del Reclutamiento por estimulación de alta frecuencia en conjunción con la tasa de estimulación.

Como la tetanizacion (APFR) es obviamente no lineal; el concepto de estimulación de alta frecuencia consigue aislar completamente el proceso de reclutamiento manteniendo constante el APFR. Este aislamiento conduce a la linealización de dicho proceso condición necesaria y suficiente para lograr graduar externamente el nivel de fuerza.

La estimulación de alta frecuencia utiliza dos estímulos independientes suministrados a las motoneuronas. La señal que produce el fondo de fuerza provee impulsos que provocan una contracción tetánica, la llamaremos SEÑAL MANEJADORA (proceso APFR). La segunda señal suministra impulsos de alta frecuencia (más de doscientos Hertz) que ocasionan una disminución en el perfil de fuerza; la denominaremos SEÑAL BLOQUEADORA (proceso de reclutamiento).

Se define efectividad de bloqueo (EB) como la relación porcentual entre la caida de fuerza durante la estimulación de alta frecuencia y la fuerza obtenida durante la tensión tetánica precedente (Figura 3). Durante el período de bloqueo no se puede lograr una atenuación completa de la fuerza disponible; siempre permanece alguna tensión residual, por tanto una EB del 100% es una hipótetica, aunque practicamente inobtenible pérdida completa en la fuerza muscular.

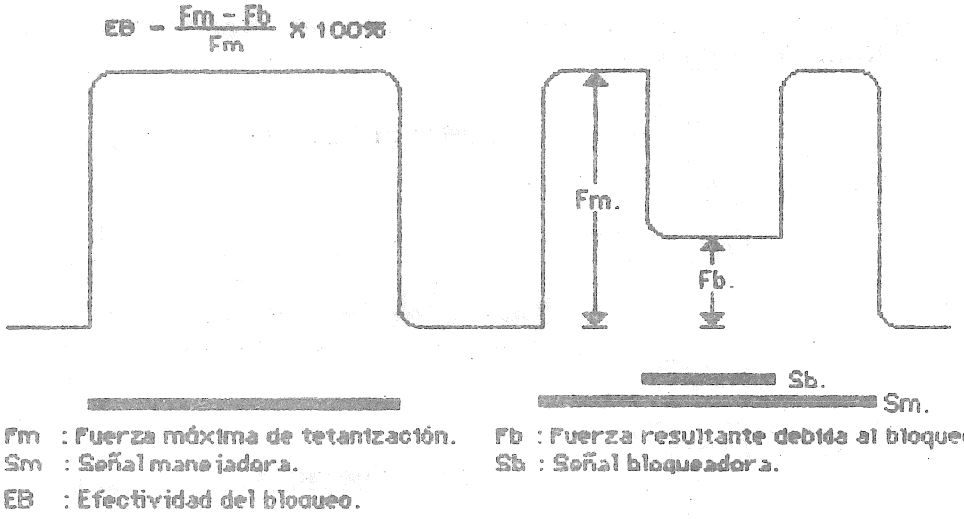


FIGURA 3 Esquema de la respuesta en fuerza a una secuencia de estimulación de alta frecuencia.

Respecto al bloqueo, existen dos hipótesis[6] que pretenden explicar la atenuación en la fuerza al aplicar estimulación de alta frecuencia. La primera de ellas (21), (25), habla de la inhabilidad temporal en la conducción nerviosa debido a una rápida depleción de la acetilcolina (ACH). La segunda hipótesis (18), más reciente, asume que el bloqueo se debe al almacenamiento de iones calcio en los túbulos transversos.

2. DESARROLLO EXPERIMENTAL

2.1 MONTAJE

Se emplearon sapos "bufos" en los cuales el músculo gastrocnemio se uso para medir la fuerza isométrica desarrollada. En el animal previamente descerebrado, se procede a cortar el tendón correspondiente. La parte libre se sujetará posteriormente al transductor de fuerza; realizada esta operación, se corta la piel en la parte posterior del animal encima del orificio urinario.

A sus costados y abriendo un poco, apareceran los dos nervios ciáticos los cuales se limpian de tejido conectivo en una distancia de varios centimetros. Luego se evita la conducción via aferente cortandose las raices de dicho nervio. Una vez lista la preparación nervio-músculo, el animal se ubica en una tablilla y se fija la parte posterior de la pata tal que el músculo al sujetarse al transductor forme un ángulo recto con el resto del cuerpo, o sea el músculo sobre el que se medira la fuerza debe estar sobre el eje vertical, evitandose así la formación de componentes del vector fuerza. El par de electrodos manejadores se situan aproximadamente en la parte media de la longitud descubierta del nervio, y de 1-2 cm. distalmente se ubican los electrodos de bloqueo. En ambos casos el electrodo más próximo al músculo debe estar conectado al terminal negativo del correspondiente estimulador.

2.2 FASES

El esquema utilizado para demostrar la factibilidad de la contracción muscular proporcional, involucra básicamente dos fases bien definidas aunque relacionadas una con la otra.

---

[6] Los autores no pretenden profundizar en este tema y dejan a consideracion del lector la validez de estas.

2.2.1 Fase uno. Se consideran dos estímulos; el manejador y el bloqueador como se observa en la Figura 4.

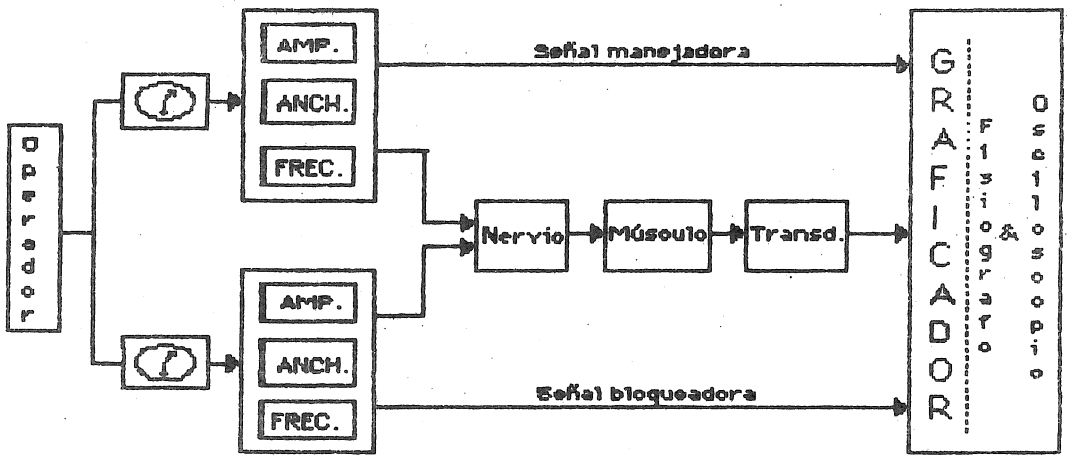


FIGURA 4 Diagrama de bloques de la fase1.

La secuencia de estimulación que se emplea es similar a la utilizada por McNeal (23), con la aplicación primero del estímulo manejador. Después que la fuerza muscular alcanza su fase lineal (en aproximadamente dos segundos), el estímulo de bloqueo es aplicado durante dos segundos (Figura 5). Una vez que se recobra la fuerza tetánica, dos segundos más tarde, el estímulo manejador se remueve.

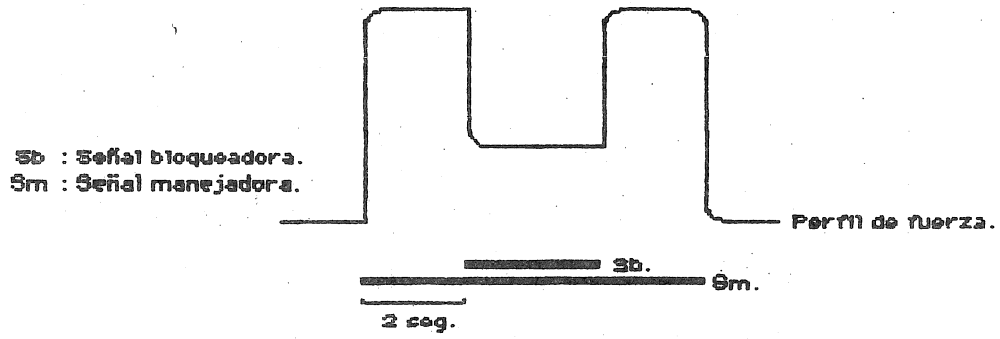


FIGURA 5 Secuencia de estimulación aplicada al nervio ciático.

Los parámetros del estímulo manejador, fijados de tal manera que minimizan la fatiga (14), son:

- ancho de pulso : 0,1 msg.
- frecuencia : 50 hertz.
- amplitud : dos veces el voltaje requerido para conseguir el umbral de excitación.

Los parámetros del estímulo de bloqueo, consistente en un tren de pulsos de alta frecuencia para la primera fase, son la variable experimental. Se determina para esta señal los correspondientes valores de amplitud, ancho de pulso y frecuencia, tales que minimicen la fatiga y se obtenga una efectividad de bloqueo aceptable.

Teniendo en cuenta la hipótesis propuesta por Solomonow (13) y considerando trabajos realizados por Tanner (16), Solomonow (12), (14), (15) donde se definen rangos de interés para la frecuencia y ancho de pulso (0,2-20 KHtz y 25-250 µseg respectivamente); se diseñan experimentos que provean valores óptimos de los parámetros que permitan lograr características de bloqueo

apropiadas.

Los experimentos se dividen en tres partes; cada una de las cuales fijara un valor para el parámetro en estudio. El protocolo utilizado involucra que se mantengan constantes dos parámetros mientras se varia el restante con miras a establecer un valor óptimo del mismo.

En conclusión, valores aceptables en los parámetros de la señal bloqueadora proporsionados por esta fase son:

- ancho de pulso : 70 µseg.
- frecuencia : 600 Hertz.
- amplitud : 2 voltios.

2.2.2 Fase dos. Al igual que en la fase uno se involucran dos estímulos; el manejador y el bloqueador (Figura 6).

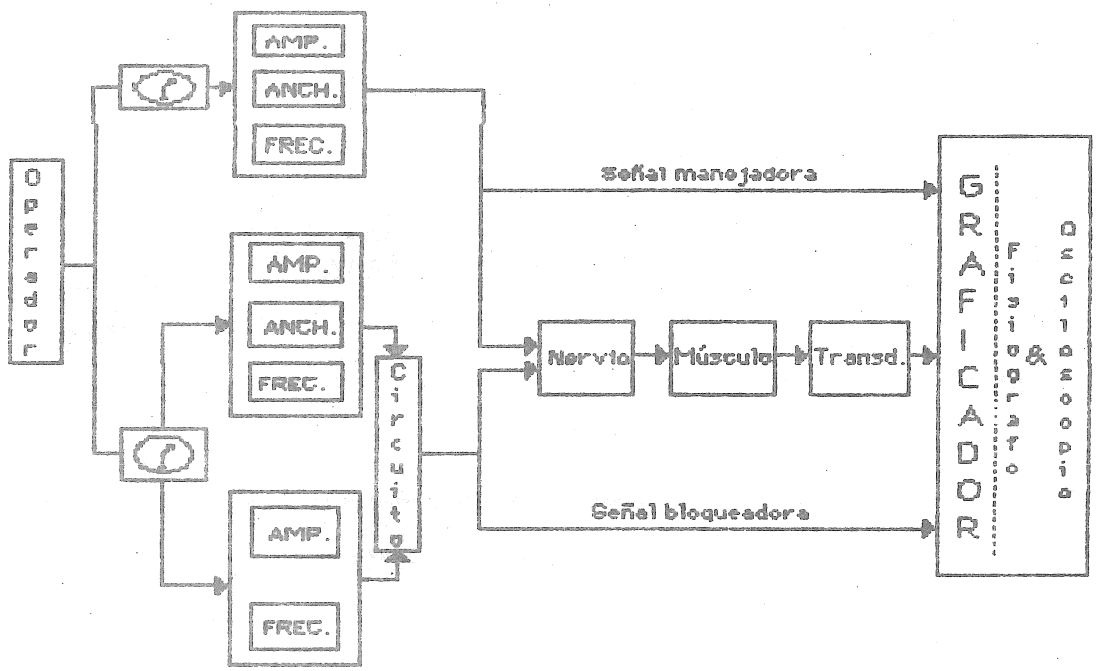


FIGURA 18 Diagrama de bloques de la fase 2.

Los valores de los parámetros para el estímulo manejador (APFR) son los mismos determinados en la fase uno. Respecto al estímulo bloqueador ahora no consiste solo en un tren de impulsos rectángulares como sucedia en le fase uno, sino que consta de una envolvente senosoidal superpuesta en un nivel DC (Figura 7).

El nivel DC se ajusta para obtener una EB del 50%. La amplitud pico a pico de la senosoidal también se ajustó experimentalmente para obtener fluctuaciones en el rango de 12,5-87,5% de la fuerza máxima.

Esta señal se logra implementando el circuito de la Figura 8, que es básicamente un sumador de las dos señales que tiene como entrada; un tren de pulsos rectángulares y una señal senosoidal.

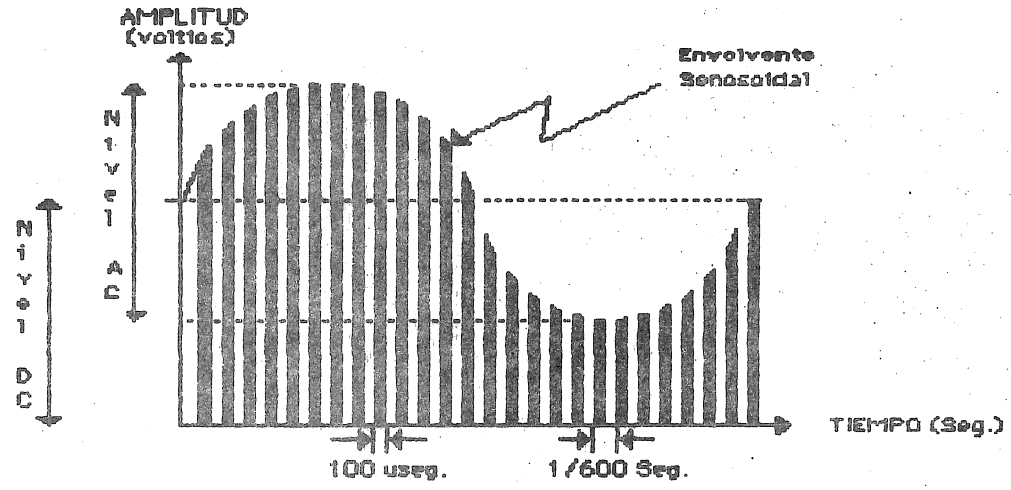


FIGURA 7 Señal bloqueadora de la fase 2.

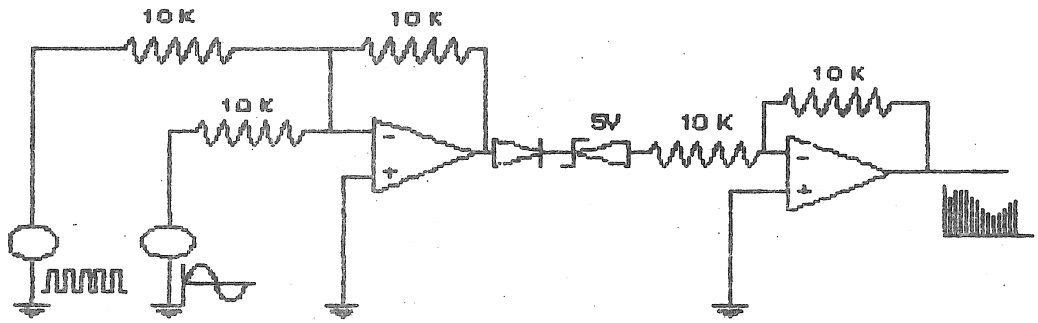


FIGURA 8 Esquema del circuito que modula la señal senosoidal mediante impulsos rectangulares.

La señal que se obtiene como salida del anterior circuito, tiene como parámetros:

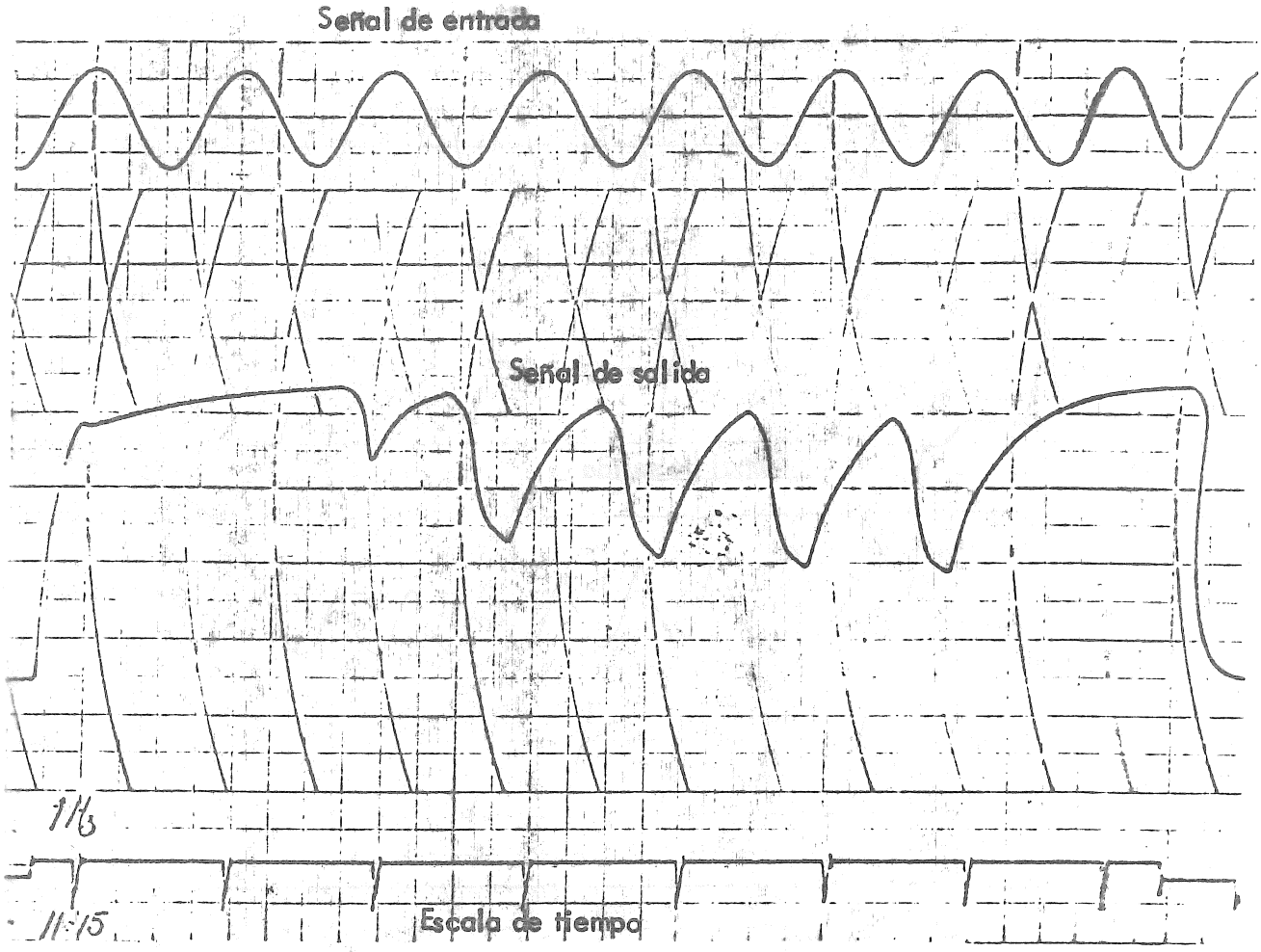
- Amplitud del pulso	2 voltios.
- Ancho del pulso	70 µsg.
- Frecuencia del tren de pulsos	600 Hertz.
- Amplitud pico a pico de la senosoidal	3 voltios.
- Frecuencia de la envolvente senosoidal	0,4-15 Hertz.

2.3 RESULTADOS

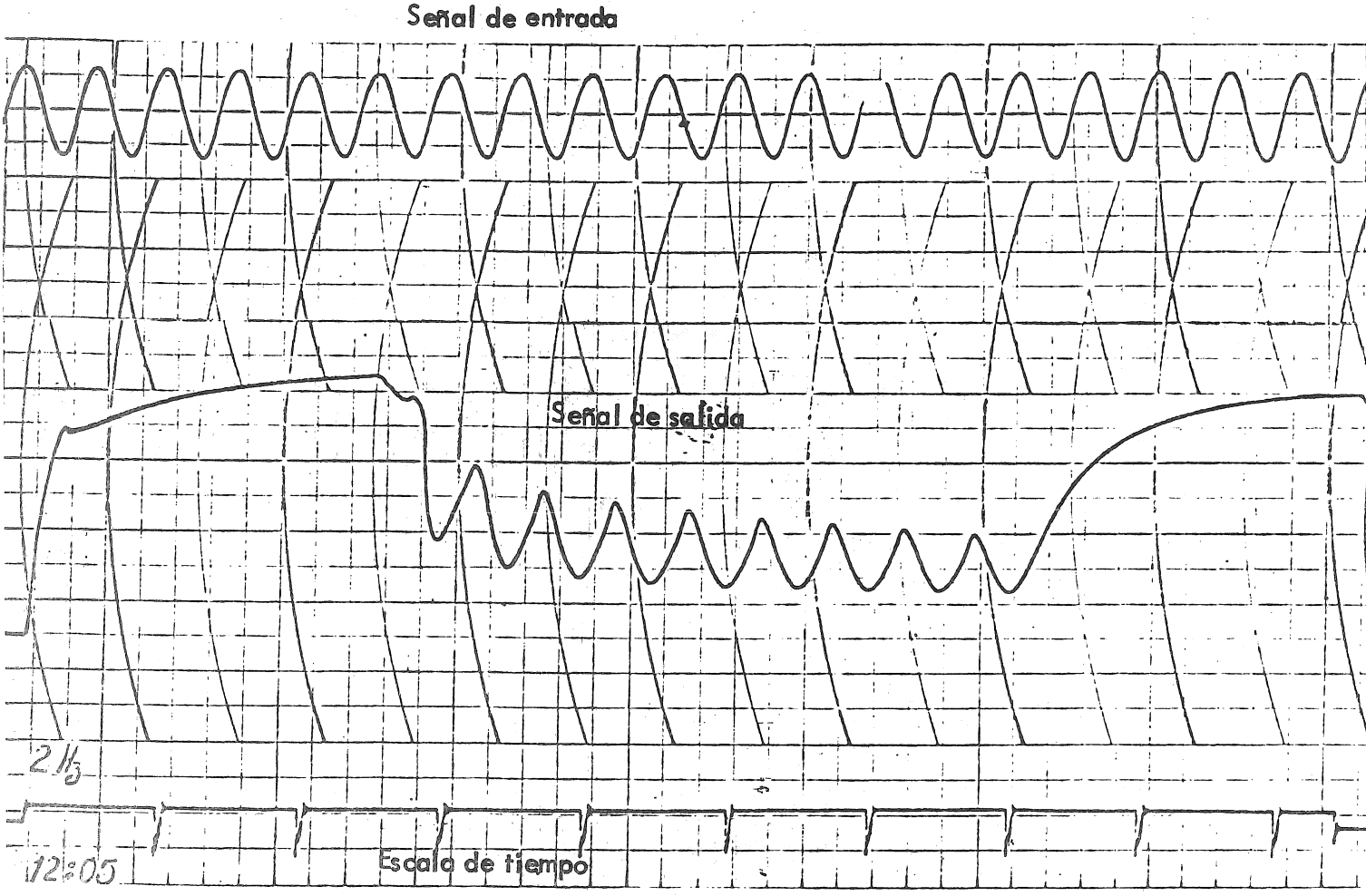
El sistema considerado consta de una entrada (amplitud de la señal bloqueadora) y una salida (fuerza ejercida por el músculo). El modelo matemático relaciona linealmente estas variables y se representa mediante una Función de Transferencia obtenida a partir del análisis de la respuesta en frecuencia del músculo (1), (3), (5).

Puesto que se utilizan los Diagramas de Bode como herramienta gráfica para la determinación del modelo, se debe tener en cuenta tanto la relación de amplitud pico a pico, como el desfase que se presenta entre la señal de entrada (medida en Voltios) y la señal de salida (medida en Kgf).

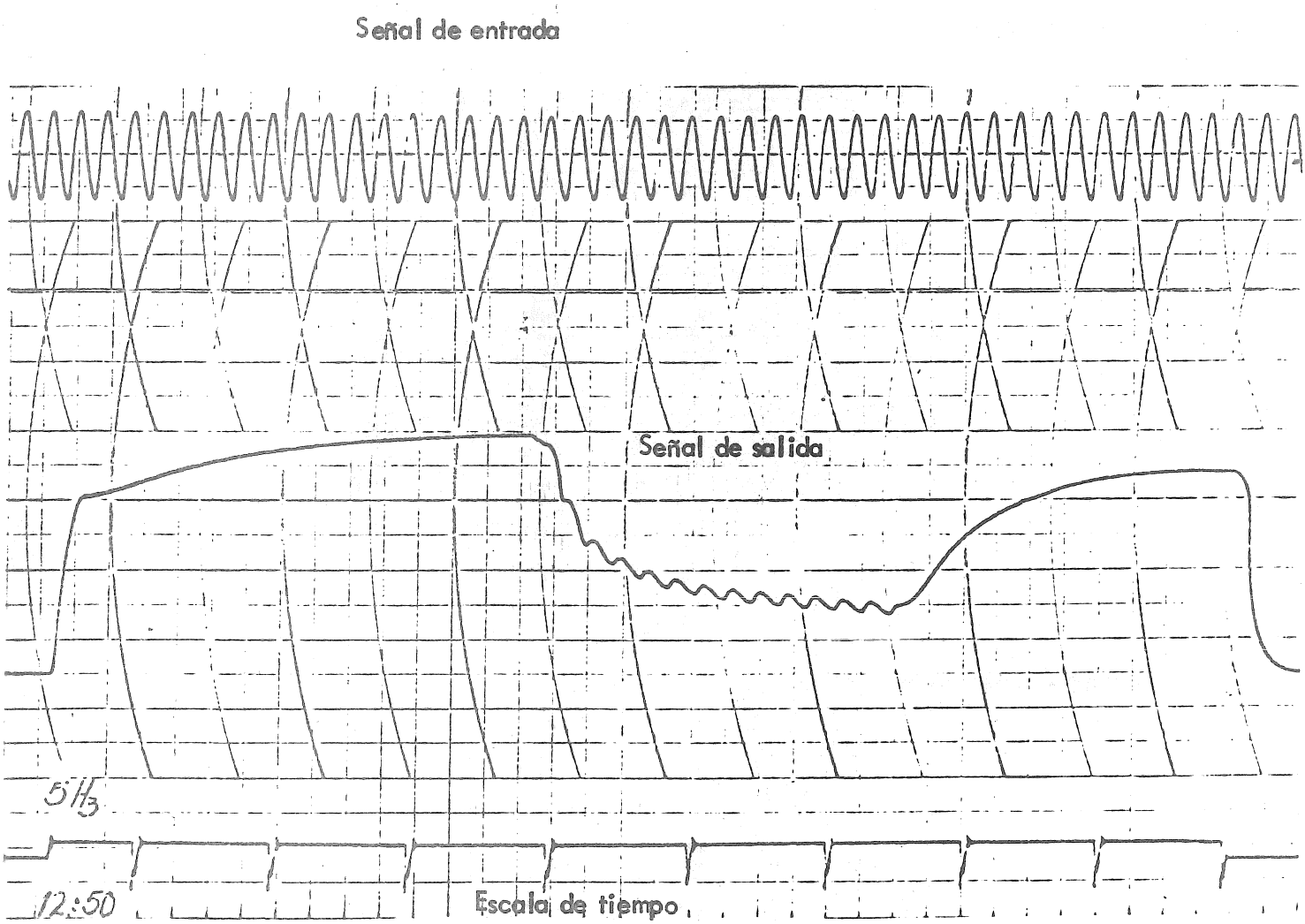
Los datos requeridos para la configuración de los diagramas, de magnitud y fase, son proporcionados por la fase dos del experimento. Algunos de los resultados que se obtuvieron son mostrados a continuación.



Respuesta en fuerza de un envolvente de un (1Hz)



Respuesta en fuerza de un envolvente de dos (2Hz)



Respuesta en fuerza de un envolvente de ánco (5 Hz)

3. FUNCION DE TRANSFERENCIA

Los datos obtenidos en la fase dos (respuesta en frecuencia); permiten hallar un modelo, logrado mediante la linealización del proceso de reclutamiento, que responde a los requerimientos de graduación de fuerza.

Se define ganancia, como el cociente entre la fuerza pico a pico (medida de los datos obtenidos en la fase dos del experimento) y la amplitud pico a pico de la señal de entrada (magnitud del nivel AC medida en voltios).

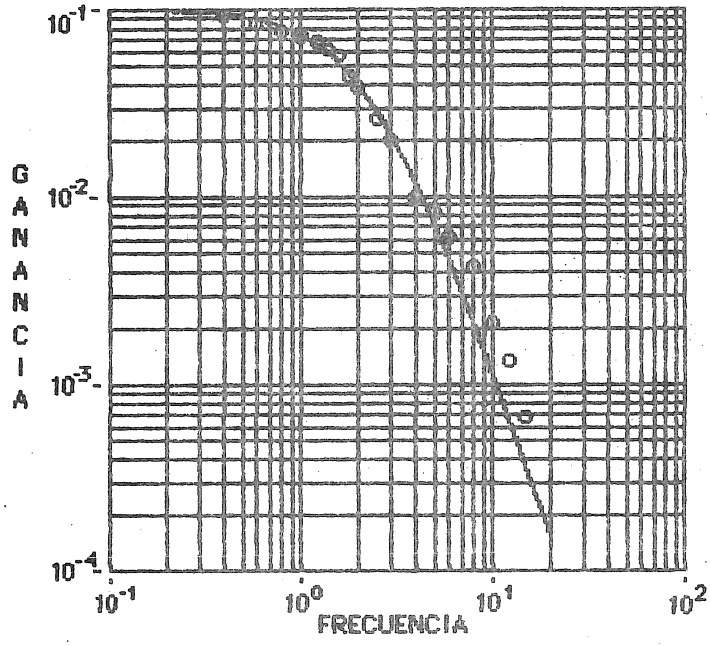


FIGURA 9 Diagrama de magnitud. La frecuencia esta en Hertz.

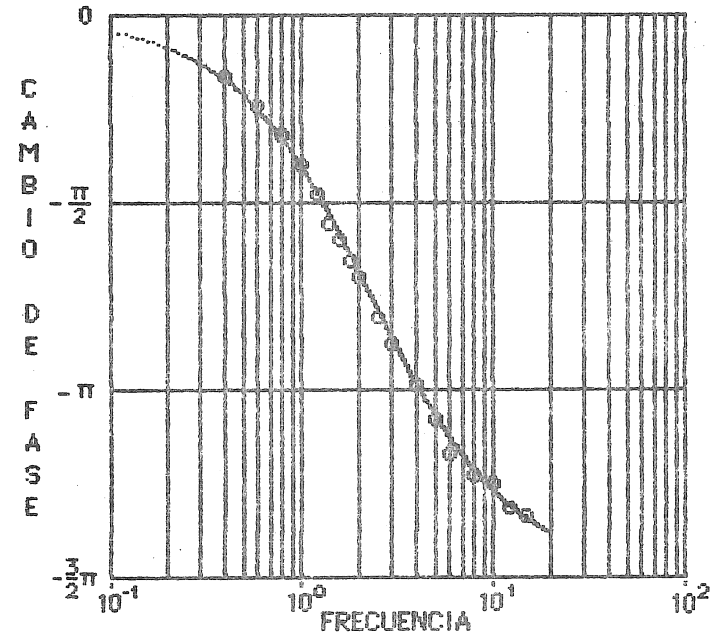


FIGURA 10 Diagrama de fase.

El diagrama de fase, especifica que la planta es de tercer orden puesto que el cambio de fase tiende a doscientos setenta grados (270°), como se muestra en la Figura 10.

El comportamiento muscular semejante a un filtro se habia detectado anteriormente (6), (12). Es importante notar que los experimentos llevados a cabo conducen a la determinación de un modelo de tercer orden, a diferencia de otras investigaciones donde se proponian modelos de segundo (6) y cuarto orden (12).

Para encontrar las frecuencias de corte se empleo el criterio de la minima suma de los cuadrados del error con base en un ajuste mayor en el diagrama de fase, pues en últimas es éste quien juega un papel preponderante.

La respuesta obtenida fué un polo doble en 1,79 Hertz y un polo sencillo en 3,94 Hertz. Por lo anterior, la función de transferencia se puede plantear como :

$$G_s(S) = \frac{0.1}{\left(\frac{S}{11.25}+1\right)^2\left(\frac{S}{24.75}+1\right)}$$

Simplificando :

$$G_s(S) = \frac{313.3}{(S+11.25)^2(S+24.75)}$$

El modelo se simulo en computador para probarlo y verificarlo. Si la entrada se hace de tipo senosoidal con una amplitud de 1.5 voltios y empleando el rango de frecuencias experimentales, se obtienen las Figuras 11 y 12, que al ser contrastadas con las obtenidas en laboratorio se observa que el error es de menos del diez porciento (10%); tanto en el nivel de fuerza, como en el comportamiento en el tiempo.

Magnitud pico a pico	Señal de entrada	3 Voltios
	Señal de salida	.17 Kgf.
Desfase	95.6 grados.	

FIGURA 11. Simulación de la respuesta a 1.4 Hertz

Magnitud pico a pico	Señal de entrada	3 Voltios.
	Señal de salida	0.15 Kgf.
Desfase	114.87 grados.	

FIGURA 12. Simulación de la respuesta a 1.8 Hertz

4. SISTEMA DE CONTROL

4.1 INTRODUCCION

Las características de todo sistema de control son estabilidad, exactitud y rapidez en la respuesta; las cuales se pueden representar mediante sus especificaciones temporales de error en estado estable, tiempo de asentamiento, porcentaje de sobrepico, etc.

En el diseño de un sistema de control se plantea un compromiso entre los diferentes requerimientos, de tal manera que se establezca la relación más eficiente (9).

Cuando un sistema no cumple con los requerimientos funcionales y debido a la imposibilidad física de reestructurarlo, se hace necesario diseñar un sistema de control para él que permita obtener dichos requerimientos. El sistema considerado tendrá una función de transferencia particular, que alterará favorablemente el sistema general.

4.2 DISEÑO

Para el músculo Gastrocnemio, que se contrae con rapidez moderada y ajuste no tan fino, se tienen las siguientes especificaciones de control que se determinan experimentalmente.

- Tiempo de Asentamiento (tss) igual a 0,3 segundos. Bajo estimulación externa el músculo se tetaniza en aproximadamente 0,3 segundos. Se considera que el tiempo de asentamiento es el mismo.

- Intervalo de confiabilidad (∞) del 5%. El ajuste no tan fino del músculo nos lleva a tomar este valor.

- Error en estado estable (ess) igual a cero. Es deseable un seguimiento en estado estable igual a cero.

- Porcentaje de sobrepico (PO) menor o igual al cinco por ciento. La respuesta transitoria no debe tener un sobrepico mayor del estipulado.

- Para evitar daños físicos en el nervio, la amplitud de la señal de control no debe exceder los diez voltios; puesto que a valores altos la respuesta del músculo se altera de tal forma que no hace confiable los valores obtenidos (experimentalmente se comprobo lo anterior).

El sistema de control real se podría representar como se muestra en la Figura 13.

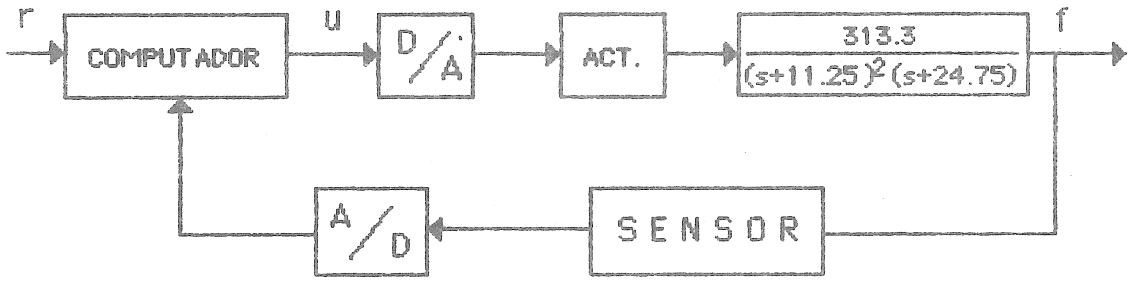


FIGURA 13 Un esquema de implementación digital.

La función de control que desempeña el computador en el esquema considerado, se representa mediante una función de transferencia, $G_c(z)$.

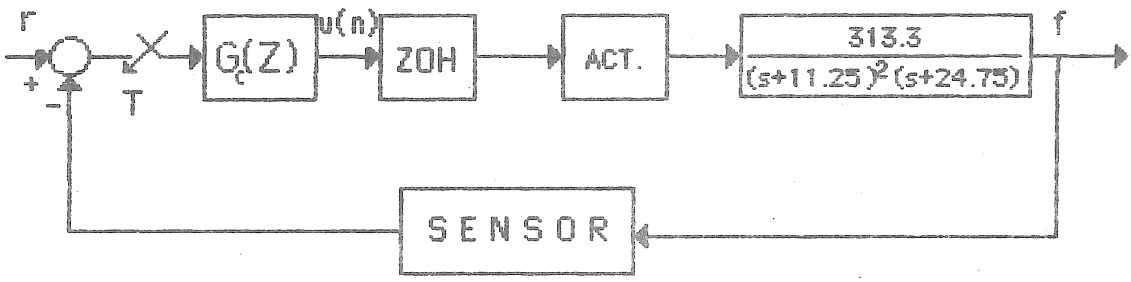


FIGURA 14 Analogia del sistema de control.

Como el sensor convierte 0,1 kilogramo-fuerza en 0,4 voltios, su función de transferencia es cuatro (Gt(s) = 4). El esquema anterior se transforma mediante el algebra de los diagramas de bloques en la siguiente figura.

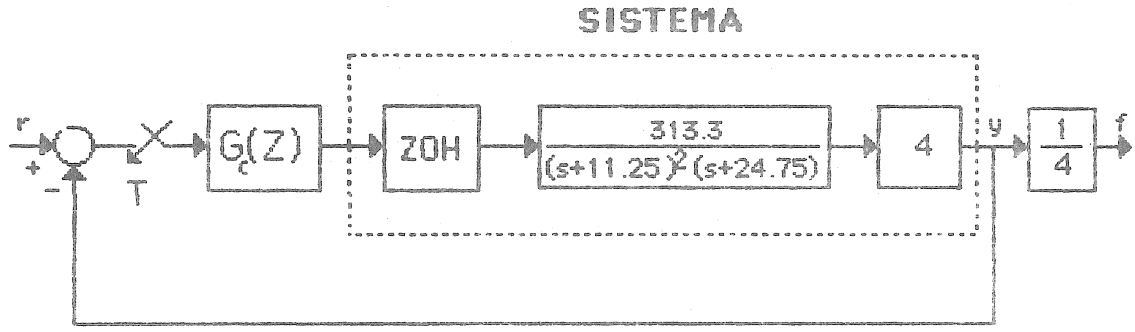


FIGURA 15 Sistema de control realimentado, donde el compensador se encuentran en serie.

Con las especificaciones y un factor de amortiguamiento de 0,7, se escoge un

período de muestreo de **0,06** seg.

Discretizando la planta mediante la técnica del ZOH:

$$G_p(Z) = \frac{Z-1}{Z} Z\left\{\frac{Gp(S)}{S}\right\}$$

Se tiene

$$G_p(Z) = \frac{0.024337(Z+0.12515)(Z+1.862339)}{(Z-0.217303)(Z-0.4996)^2}$$

El sistema discreto es mostrado a continuación.

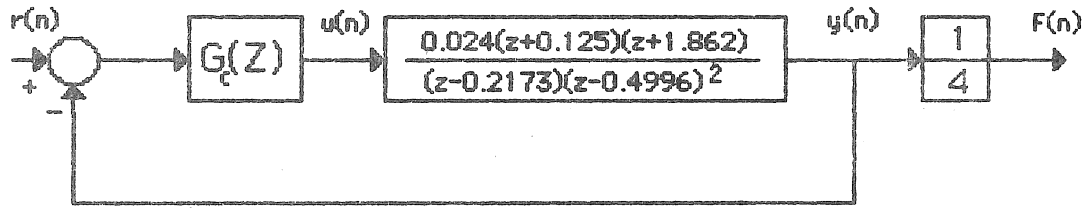


FIGURA 16. Sistema de control discreto.

Si se asume Gc(z) = 1, el sistema por si solo no cumple con los requerimientos estipulados con antelación.

El sistema en lazo cerrado se debe asemejar a un sistema de segundo orden, donde la ubicación de los polos dominantes en el círculo unitario del plano z, se ve afectada por el cero de la planta que se encuentra fuera de él.

Observando la respuesta de un sistema de dos polos en el círculo unitario y un cero fuera de él, se determino que los polos conjugados dominantes deben estar situados aproximadamente en 0,3 +/- j0,3, lo que asegura el cumplimiento de los requerimientos.

Para que a un paso el error en estado estable sea cero (ess = 0), el compensador debe ser de tipo uno o sea tener un polo en uno. Igualmente, manipulando el Root-Locus, se especifica que el compensador tendrá tres ceros y un polo que cancelan los polos y el cero (ubicado en el círculo unitario) de la planta. Estas cancelaciones se basan en la exactitud de las mediciones.

El otro polo del compensador se localiza de tal manera que el Root-Locus pase por el punto deseado, para ello debe cumplir con la regla del ángulo que dice "la suma de los ángulos en lazo abierto debe ser igual a $(2n+1)\Pi$ si $k \geq 0$ donde n es entero". De esta manera el polo se situa en -0,344964 (z+0,344964).

Por consiguiente Gc(z) es un sistema de tercer orden con tres polos y tres ceros en :

$$G_c(Z) = \frac{9\,(z-0.217303)(z-0.4996)^2}{(z+0.12515)(z+0.344964)(z-1)}$$

Por tanto, el sistema discreto de control en lazo cerrado propuesto es

mostrado en la Figura 17

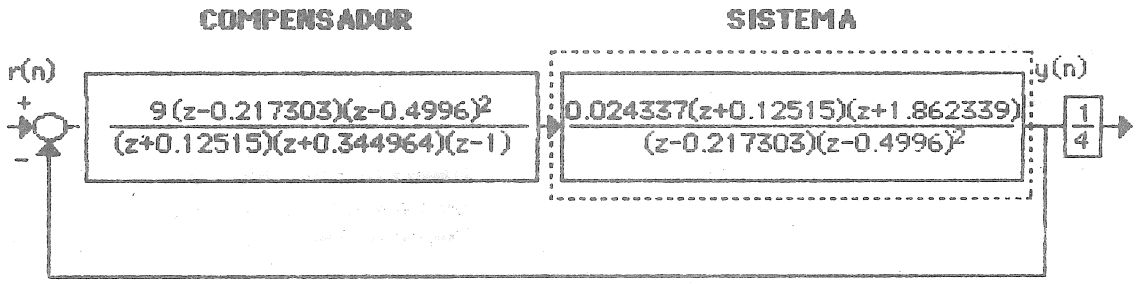
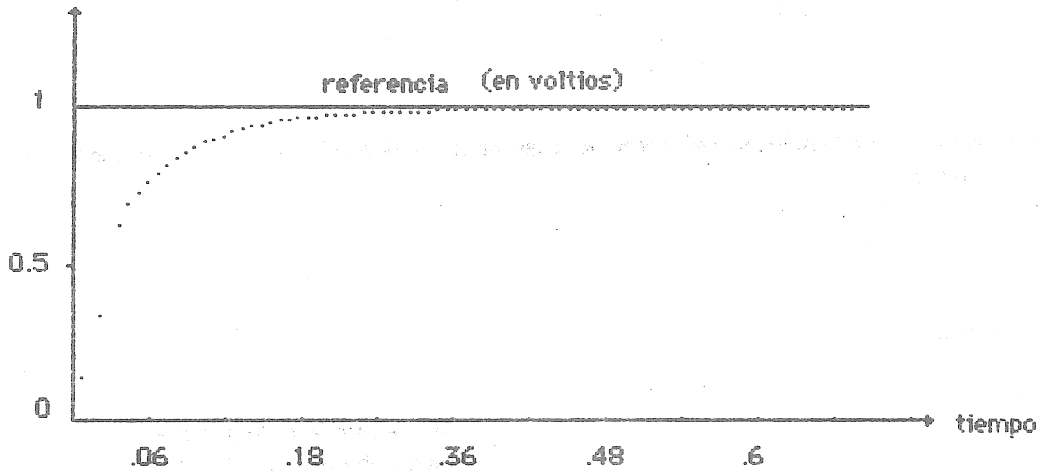


FIGURA 17. Sistema de control discreto para el músculo esquelético.

La respuesta en tiempo es mostrada en la Figuras 18. El análisis de esta gráfica muestra que el sistema responde a los requirimientos y las entremuestras corroboran el comportamiento del sistema en mención.



TIEMPO DE ASENTAMIENTO : 0.18 Seg.
PORCENTAJE DE SOBREPICO : 0

FIGURA 18. Respuesta en tiempo del sistema de control.

5. SIMULACION

Para comprobar que el sistema de control digital cumple a cabalidad los requerimientos para los que fué diseñado, se simulo en computador su comportamiento y los resultados se contrastaron con los obtenidos experimentalmente para probar su validez.

La simulación se basa en la representación del sistema por medio de las variables de estado discretas. La función de control discreta se paso a una ecuación de diferencia, que se implementa posteriormente a nivel de software.

Los resultados obtenidos por dicha simulación son mostrados en las Figuras 19, 20 y 21.

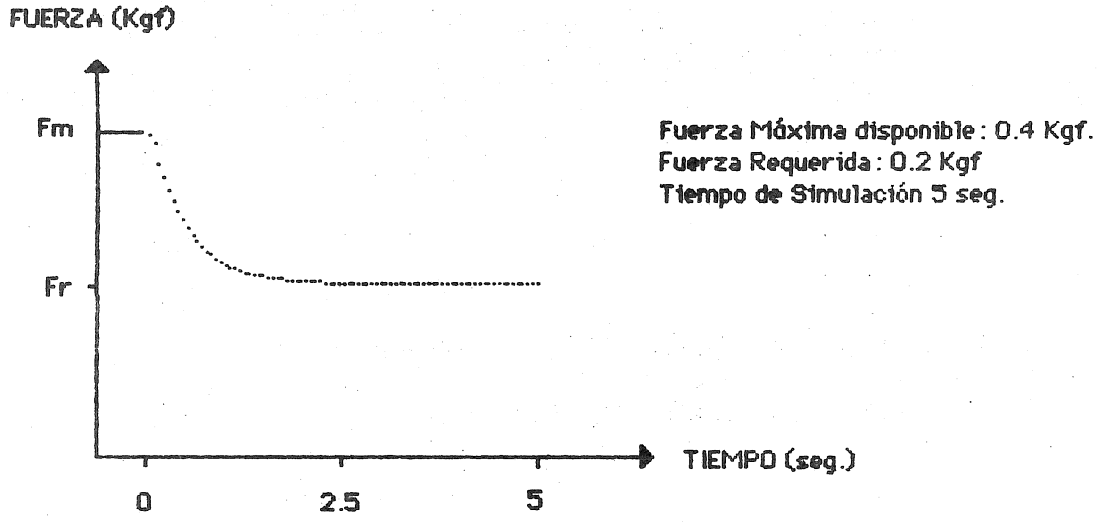


FIGURA 37. Simulación de la respuesta del sistema a un control ejeroido por el computador. La fuerza requerida es 0,2 Kgf.

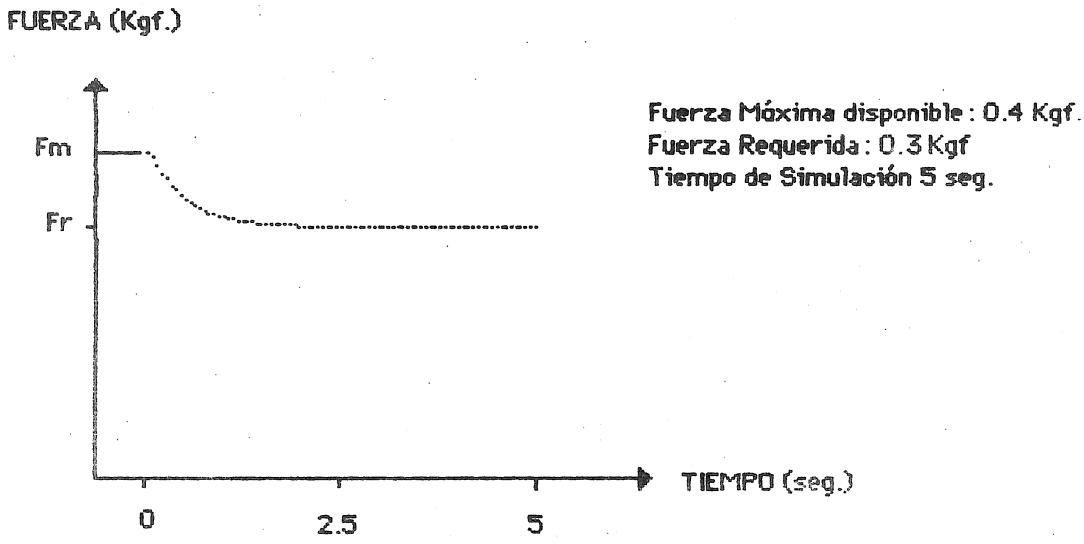


FIGURA 38. Simulación de la respuesta del sistema a un control ejercido por el computador. La fuerza requerida es 0,3 Kgf.

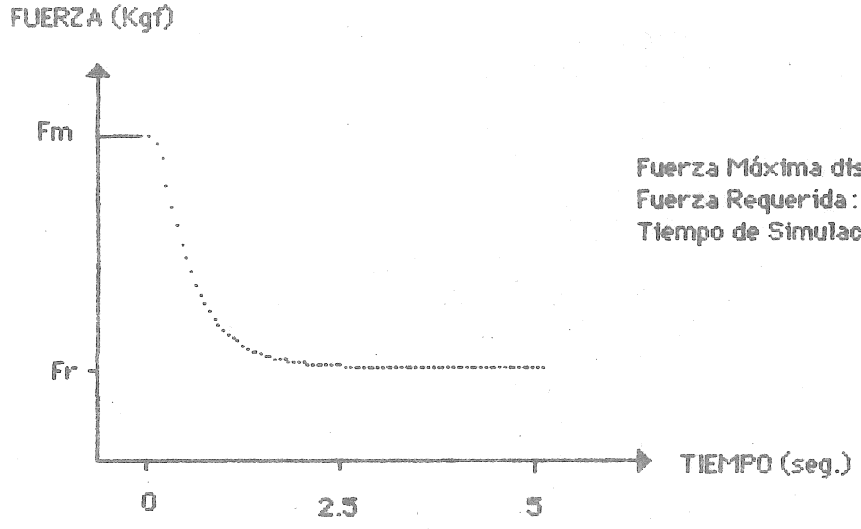


FIGURA 39. Simulación de la respuesta del sistema a un control ejercido por el computador. La fuerza requerida es cero.

Es esencial el conocimiento previo del máximo nivel de fuerza que pueda ejercer el músculo en estudio. Una vez determinado este, el control es asumido por el computador que de acuerdo con el nivel de fuerza solicitado, cumple con su función de control.

Los resultados de la simulación preveen un control adecuado del nivel de fuerza muscular, lo cual hacen factible su implementación real para comprobar si efectivamente esta técnica de control es adecuada.

Posteriores investigaciones verificarán la presente investigación.

BIBLIOGRAFIA

(1) ELGER, Olle Ingemar. Control system theory. Tokyo : Mc Graw-Hill, 1967.
(2) GUYTON, Arthur. Tratado de fisiología medica. 6 ed. Madrid, España: Interamericana, 1984. p. 749-756. ISBN 84-7242-351-4.
(3) HARRISON, Howard and BOLLINGER, John. Controles Automaticos. 3 ed. Mexico : Trillar, 1979. ISBN 968-24-0120-8.
(4) HENNEMAN, E and HOUK, J. Neural control of movement and posture. En: MOUNTCASTLE, Vernon. Medical Physiology. Saint Louis, MO : Mosby, Vol. 1, part V, 1974.
(5) KUO, Benjamin. Sistemas automáticos de control. 2 ed. Mexico : Continental, 1970.
(6) MANNARD, Allan and STEIN, Richard. Determination of the frequency response of isometric soleus muscle in the cat using random nerve stimulation. En: Journal of Physiology (London). Vol. 229, (1973); p. 275-296.
(7) MILNER-BROWN, H; STEIN, R and YEMM R. Changes in firing rate of human motor units during linearly changing voluntary contractions. En: Journal of Physiology. Vol. 230, (1973); p. 371-390.

(8) NASIR, Ahmed and NATARAJAN, T. Discrete-time Signals and Systems. 1 ed. RESTON, Virginia : Printice-Hall company, 1983.
(9) NOTAS CURSO DE CONTROL. (1986 : Bogotá). MELUK, Francisco. Universidad Nacional de Colombia.
(10) PARTRIDGE, Lloyd. Modifications of neural output signals by muscles : a frequency response study. En: Journal of applied Physiology. Vol. 20, (1965); p. 150-156.
(11) PERSON, R and KUDINA, L. Discharge frequency and discharge pattern of human motor units during voluntary contraction of muscle. En: Electroencephalography and Clinical Neurophysiology. Vol. 32, (1972); p. 471-483.
(12) SOLOMONOW, Moshe. External Control of the Nueromuscular System. En: IEEE Transactions on biomedical Engineering. Vol. BME-31, No 12 (dec, 1984); p. 752-763.
(13) SOLOMONOW, Moshe et al. A new technique for functional neuromuscular stimulation. p. 47-62. En: TOMOVIC, R. Advances on External Control of Human Extremities. Belgrade, Yugoslavia : Yugoslav Committee for Electronics and Automation, 1978.
(14) _____. Control of muscle contractile force through indirect high frecuency stimulation. En: American Journal of Physical medicine. Vol. 62, No. 2 (1983); p. 71-82.
(15) _____. Fatigue considerations of muscle contratile force during high frequency stimulation. En: American Journal of Physical Medicine. Vol. 62, No. 3 (1983); p. 117-122.
(16) TANNER, J. Reversible blocking of nerve conduction by alternating current excitation. En: Nature. Vol. 195, (aug 1962); p. 712-713.

PROYECTO PILOTO DE INFORMATICA PARA EL MANEJO DE LOS IMPUESTOS EN COLOMBIA

AUTORES:

Alfredo Hernández Cifuentes.
Esperanza Bedoya Vargas.

DIRECCION

MINISTERIO DE HACIENDA Y CREDITO PUBLICO
Carrera 8 No. 6 - 40

XIII CONFERENCIA LATINOAMERICANA DE INFORMATICA

RESUMEN

El principal usuario del Centro de Información y Sistemas del Ministerio de Hacienda y Crédito Público es la Dirección de Impuestos Nacionales. Las aplicaciones existentes en este momento han estado en funcionamiento por cerca de dos décadas y el sistema nunca ha estado 100% completo. El diseño actual de las aplicaciones, la tecnología utilizada y las herramientas con que cuentan los programadores del CIS, fueron las principales razones para que se iniciara un proyecto que implicaría el total rediseño del sistema a nivel informático, operativo y legal.

Entre Abril de 1.985 y Agosto de 1.986, en una estrecha relación entre el CIS y la DIN y a través de prototipos rápidos se hizo la especificación de requerimientos y se establecieron los puntos principales del diseño, los cuales se experimentarían a través de una "Unidad Piloto".

En este momento se están empezando a llevar a un ambiente real las primeras etapas del proyecto por medio del "Piloto Informático" a través de lo que se ha denominado "Ambiente de Entrada de Datos" debido a que sus procesos son los del manejo inicial de documentos. Este dejará el camino para implantar las otras etapas permitiendo así el desmonte paulatino de los actuales sistemas.

INTRODUCCION

La Direcciòn General de Impuestos Nacionales (DIN) cuenta con 29 Administraciones regionales y 894 recaudaciones las cuales tienen como funciòn primordial el control del cumplimiento de las obligaciones tributarias, tanto de los contribuyentes como de los responsables de recaudo. El cubrimiento de la Direcciòn de impuestos es de aproximadamente 3 millones de contribuyentes los cuales se han disminuido en dos terceras partes debido a las nuevas reformas en el sistema tributario. Fundamentalmente, se administran tres tipos de impuestos: renta, ventas y retenciones en la fuente.

El Centro de Informaciòn y Sistemas (CIS) administra el equipo de computaciòn del Ministerio para el manejo y servicio de la red de teleproceso; este equipo (I.B.M. 4381) tiene conectadas aproximadamente 1.000 terminales en el paìs, utilizadas por la DIN para la atenciòn de los contribuyentes y por el CIS para la trascripciòn masiva de datos a nivel central y en 21 regionales. Adicionalmente, existen otros dos computadores I.B.M. 4341 de los cuales uno es utilizado para los procesos "Batch" y el desarrollo de "software"; el otro denominado 'Infocentro' es utilizado para proyectos muy puntuales y de cierta urgencia.

Dada la organizaciòn actual, el CIS tiene centralizado el desarrollo del "software", que es realizado por un grupo de 30 programadores y analistas, los cuales utilizan como herramienta primordial el lenguaje P.L.1. Es importante anotar que el usuario bàsico del equipo, aproximadamente en un 80 %, es la Direcciòn y Administraciones de Impuestos Nacionales.

El proceso tributario consta de tres partes a saber:

A. Determinaciòn: consiste en la presentaciòn anual de una declaraciòn por parte del contribuyente.

B. Fiscalizaciòn: tiene como propòsito la identificaciòn de errores involuntarios o con el fin de evadir.

C. Recaudaciòn: tiene como propòsito el cobro de los pagos de las obligaciones originadas por la determinaciòn y la fiscalizaciòn.

Para el buen desarrollo de los tres procesos se lleva por cada contribuyente una cuenta corriente, donde se registra el valor del tributo a cargo, los pagos o abonos a la obligaciòn, a los intereses y sanciones, etc. y que permite determinar quièn es moroso y quièn no.

El denominado Proyecto Piloto de Impuestos busca rediseñar el sistema actual, que lleva màs de 20 años en funcionamiento, utilizando tecnologìa de "software" y de "hardware" reciente que permita al Ministerio y especialmente a la Direcciòn de Impuestos dar un servicio àgil y eficiente a sus contribuyentes, corregir errores estructurales del sistema, completar el "software" aplicativo y mejorar el nivel de recaudo, todo esto con mayor control y seguridad.

I. ANTECEDENTES DEL PROYECTO

A. Historia.

El Proyecto Piloto, parte de un estudio hecho, entre los meses de Abril y Diciembre de 1985, del sistema actual. El estudio se dividió en dos partes. La primera desde un punto de vista Tecnológico e Informático, y la segunda desde un punto de vista sistémico. Para el estudio se tuvo en cuenta el comportamiento del ambiente tributario y de la automatización en los últimos veinte (20) años. De esta manera, se encontraron inconvenientes de oportunidad de la información, rigidez e ineficiencia, que exigían un replanteamiento de todo el sistema. La misma configuración del equipo es en este momento obsoleta y costosa. Como resultado en primera instancia se planeó el rediseño de la cuenta corriente de renta, donde se pretendía profundizar más en el problema informático y tecnológico que en el problema administrativo.

Luego durante los meses de Enero a Agosto del 86, en estrecha colaboración con la DIN y un asesor internacional, se logró profundizar en el problema administrativo y conceptual de los procesos tributarios de determinación y recaudación, y se replantearon todos los procesos desde tres puntos de vista principalmente: Informático, Operativo y Legal. En esta nueva fase se redefinieron la mayoría de los procedimientos administrativos para el control de las obligaciones tributarias lo cual implicó el total rediseño de lo que existe actualmente, completarlo con los que aún no se habían diseñado, y la selección de los recursos tanto de "hardware" como de "software" apropiados y acordes con la evolución tecnológica.

B. Sistema Actual.

Hoy en día conviven dos situaciones diferentes. Una manual donde se registran las diferentes acciones de los contribuyentes y del estado en unas tarjetas, elementos del kardex de cuenta corriente manual, y la otra automática, donde el computador almacena y controla, de la misma manera que para el sistema manual.

La mayoría de las aplicaciones existentes son para recaudación en primer lugar y para determinación en segundo lugar. En el área de fiscalización son pocas las que se tienen debido a que la DIN está atrasada en este aspecto.

Entre las principales aplicaciones del sistema automático, podemos mencionar las siguientes:

Trascripción de declaraciones de renta y ventas. Digitación. corrección, validación, conformación y actualización de los pagos realizados en las administraciones sin teleproceso y en los bancos.

Servicio en línea de atención al contribuyente que incluye liquidación de deuda e intereses, pagos, expedición de Paz y Salvo, consulta e historia de la cuenta. Programas de corrección a cuenta corriente y correcciones al nombre y la dirección.

C. Razones para el cambio.

Las principales razones por las cuales en el año 86 se llegó a la conclusión de que era necesario un cambio en el ambiente informático son las siguientes:

1. Administrativas.

- **Algunas administraciones muy grandes,** lo cual afecta los controles, la operación y la administración.

- **Demasiados datos y poca información** por lo cual la oportunidad, el manejo y el control son incompletos y poco satisfactorios.

- **Divorcio entre lo manual y lo automático.** No existe comunicación efectiva entre las partes que se llevan manualmente y las automáticas.

- **No existen parámetros para estadísticas.** No se tiene una definición paramétrica de lo que deberían ser.

2. Tecnológicas.

- **Altas inversiones en equipos y pocas en recursos humanos y "software".**

- **Excesiva centralización,** una falla a nivel central repercute en toda la red, hay mayor congestión en la unidad central.

- **Dependencia de un proveedor** lo cual ata el desarrollo a una sola tecnología, principalmente en el "software".

- **Tiempo requerido para desarrollo, prueba y puesta en marcha del "software" es excesivo.** El sistema nunca ha estado completo.

- **Herramienta para el desarrollo** es bastante **ineficiente,** no se cuenta con ayudas de programación. por tanto hay un atraso constante ante los nuevos y cambiantes requerimientos de la DIN.

3. Diseño y concepción del sistema.

- No existe un dígito de chequeo al NIT, por tanto la validación se hace contra el nombre, lo cual genera gran número de inconsistencias.

- El actual diseño es incompleto por tanto se ha ido remendando a medida que las necesidades lo exigen, lo cual lo hace inmanejable, complejo, frágil y rígido.
  No está, ni nunca ha estado 100% completo.

- **No es general.** Cada vez que se crea un nuevo impuesto, genera problemas de diseño y programación que toman varios meses e inclusive años.

- No es integro. Los diferentes módulos que lo componen no están encadenados. Muchas veces es necesario salir totalmente de un ambiente para efectuar transacciones en otro, con el consiguiente retardo para el contribuyente.

- No es seguro. El "software" utilizado, el diseño incompleto y la integridad, además de todos los factores ya vistos hacen que sea fácil de intervenir.

4. Operativas.

- **Contabilidad manual.** Hasta el momento no existe ninguna conexión entre el sistema y la contabilidad, a no ser los listados que son enviados a la DIN.

- **No es auditable.** Con las cifras de control que se obtienen, no es posible conocer cuál es el proceso que ha seguido determinado documento. Tampoco existe un histórico de los cambios hechos a los programas.

- **Procedimientos desordenados.** Las pocas reglas que existen, sobre la forma como se debe manejar la información no son aplicadas correctamente.

- Se pierde relación documento físico-sistema, una vez el documento ha sido digitado, el paquete en el que inicialmente viene se rompe.

- No existe un control sobre lo que entra, se desconoce el proceso por el cuál un documento entró en un momento dado al sistema.

II. COMO SERIA EL NUEVO SISTEMA TRIBUTARIO

Este capítulo que podría ser muy extenso se resume en cinco puntos principales para dar una idea global del Nuevo Sistema Tributario. En un nivel general tenemos la filosofía que se ha seguido para el desarrollo del proyecto y algunas reformas administrativas importantes. Un nivel operativo resalta los puntos que tienen mayor incidencia sobre la situación actual; se ha querido dejar un nivel de documentos, debido a la importancia que tiene la forma como se han de manejar. Se ha resaltado un punto sobre control por que una de las principales actividades de impuestos involucra el manejo de dinero. La consistencia del sistema se resalta principalmente por la razón de que algo tan importante y que al principio puede parecer natural, no se da en el actual.

A. Nivel General.

Como filosofía para el acceso de información se han escogido las **Bases de Datos Relacionales** las cuales cuentan con un rápido acceso que además es seguro.

Administraciones de aproximadamente cincuenta mil (50.000) contribuyentes. Existen algunas administraciones muy grandes que generan gran acumulación de papeles y se vuelven inmanejables en todo sentido. El diseño contempla manejo de unidades modulares que implican tiempos de procesamiento más cortos y por tanto permiten obtener resultados a tiempo.

Es muy importante mantener los últimos cambios de **Legislación.** de forma que se pueda hacer consulta de ésta en cualquier momento. Así mismo la información general de los contribuyentes se tendrá recopilada en el archivo RUT (Registro Unico Tributario), que permitirá un manejo general; de aquí se obtendrá la información para la emisión de rótulos y para todo lo que sea necesario enviar a aquellas personas que tengan obligaciones con la Administración de Impuestos.

Sistema Distribuido, tanto en equipos como en datos. Se montará una red, con computadores en las principales ciudades, para atender necesidades locales y manejar los datos de la zona. A nivel central se llevaría una base de datos nacional con información consolidada y duplicada de los contribuyentes, para propositos de auditoría, seguridad y control.

Cada una de las personas que tenga acceso al sistema, deberá estar registrada en "Usuarios" para lo cuál se establecerán diferentes niveles de entrada, así como diversas prioridades.

La puesta en práctica de las sugerencias produciría un cambio estructural profundo de la administración tributaria. Es posible preveer que estos cambios marcarían el rumbo por más de una década.

B. Nivel Operativo

Se grabarán los documentos tal y como el contribuyente los presenta. El proceso de corrección de errores, al momento de la validación, sólo tiene que corregir los errores de transcripción (Digitación). Otros procesos serán los encargados de reclamar al contribuyente las imprecisiones que éste haya cometido en sus declaraciones.

Definir un **periodo máximo para el procesamiento** y validación total de las declaraciones y pagos. No tiene sentido invertir en equipos muy poderosos y eficientes si no se los puede alimentar a tiempo con los datos. Debe lograrse un equilibrio entre la capacidad del equipo y la periferia que lo alimenta.

Red Bancaria. Efectuar la recepción de pagos. Trasladar la recepción de todo tipo de documento. Conectar los computadores de los bancos, con los del CIS.

Producción de estadísticas. Las estadísticas formarán parte del sistema, y se tendrán en cuenta ciertos parámetros bajo los cuales se acumularán absolutamente todos los renglones del formulario. Este archivo de estadísticas se actualizará **diariamente** con los datos de la captura básica y en el momento en que se termine la radicación general se completarán los renglones que hagan falta.

Fiscalización. Se diseñarán métodos eficientes de fiscalización, partiendo del hecho que el ambiente será congruente y la información estará a tiempo y completa. Para efectos de la fiscalización se piensa utilizar al máximo la información exógena que la DIN tiene derecho a exigir y que en el momento no está en condiciones de utilizar.

Cobranzas. Se conocerá el punto en el cuál se encuentra cada proceso de cobranza. Por medio de un **archivo de morosos**, se tendrán todas aquellas personas que no están a paz y salvo con la Administración de Impuestos.

Se asume que, para el mediano y largo plazo, es conveniente que:

a) El paz y salvo desaparezca.

b) Todas las declaraciones y pagos entren por la red bancaria y eventualmente también por otras Instituciones Financieras.

En el corto plazo es necesario:

a) Los documentos formen paquetes que no se rompan.

b) Se genere una carátula para cada paquete que sirva como identificación del mismo.

C. Nivel de Documentos

Paquetes como unidad única y no divisible.
Cada 100 documentos aproximadamente formarán un paquete que llevará una carátula con algunos datos de control, por ejemplo: Total Impuesto a pagar, total Retenciones, total del Anticipo. Este proceso se sigue en todos los lugares donde haya recaudo o recepción de documentos es decir en las administraciones de impuestos, en las recaudaciones y en los bancos.

Este punto además de darle seguridad a la organización permite su auditabilidad.

Conocimiento de la ubicación física de cada paquete.
Existirá una "Biblioteca de documentos" y todos los trámites estarán automatizados. Cuando por una causa extrema se necesite un documento original, se podrá, a través del computador, saber donde se encuentra. Así se exige un orden en las operaciones y una relación entre los procesos manuales y automáticos.

Menos datos y más información. Disminución del número de renglones en el formulario lo cual produce un menor tiempo de digitación y por tanto hay actualización, a tiempo, de la cuenta corriente del contribuyente para hacer Fiscalización debidamente. Sólo se cuenta con la información necesaria para trabajar eficientemente.

Anexos no serán exigidos mientras no sea necesario. El contribuyente los guardará en su casa, así la administración no se llenará de documentos innecesarios.

D. Control.

Totalmente auditable desde el momento de la recepción. En todo momento se sabe quien está trabajando y si tiene permiso para hacerlo, además todos los mecanismos manuales que se llevan, son conocidos y grabados. Esto permite que se pueda entrar por cualquier punto que se considere importante de controlar.

Absolutamente todas **las acciones y sus responsables quedan grabadas en el sistema.** Los documentos entran tal y como el contribuyente los ha presentado.

Sistema totalmente integrado relacionado en todos sus módulos. Se tendrán todas las opciones del sistema encadenadas.

E. Consistencia.

Analogía en el tratamiento de los diferentes impuestos existentes y los que surgan en un futuro desde el punto de vista de proceso, Así cuando surga un nuevo impuesto no hay traumas operativos ni informáticos.

Contabilidad como parte integral del sistema. Existirá una actualización diaria de la contabilidad. Hoy en día la contabilidad es manual y nunca cuadra con los datos del sistema automático.

Módulos especializados en corregir inconsistencias. Se busca crear mecanismos que disminuyan la entrada de inconsistencias a la cuenta corriente, y para ello se siguen ciertos pasos previos a la confirmación de la cuenta corriente. (Validadores, Corrección aritmética, etc.). Una adición importante es el dígito de verificación, porque la validación contra el nombre que se hace hoy en día no es totalmente confiable. El diseño de los módulos para los bancos disminuirá los errores producidos por éstos y por tanto la actualización de la cuenta corriente va a ser más efectiva.

Con un excelente sistema de corrección de inconsistencias y de actualización de estadísticas y contabilidad, la información que se podrá obtener será veraz y confiable. Por tanto dará herramientas a los fiscalizadores y cobradores para efectuar su labor de una manera eficiente y lógica, características dificiles de lograr hoy en día.

III. ESTRATEGIA DE DESARROLLO

Del estudio y las evaluaciones hechas en el año 1.986 se llegó a un diagnóstico final que determinó el perfil de lo que sería el nuevo sistema tributario colombiano. El diseño incluye todo lo existente hoy en día y todo lo que aún no se ha diseñado a fin de cubrir el proceso de determinación, el de fiscalización y el de recaudo. Para ello se han seguido las siguientes estrategias:

A. Prototipos Rápidos.

Se han utilizado los prototipos rápidos en la especificación de requerimientos del usuario, porque se consideran una excelente ayuda para conocer en detalle las particularidades de un ambiente de "software", experimentar con la interfaz y como herramienta interactiva para el desarrollo. Permiten además tener una visión del futuro sistema, efectuando los cambios o adiciones en las primeras etapas del proyecto y no al final, cuando un cambio puede implicar una reestructuración total.

B. Unidad Piloto.

La estrategia para llevar los prototipos al ambiente real se había definido a través de una "Unidad Piloto" en la cuál se iban a manejar inicialmente los "Grandes Contribuyentes". Esta Unidad Piloto estaría situada en el edificio San Agustín de la ciudad de Bogotá y permitiría poner en experimentación tanto el aspecto tributario como el informático. Sin embargo se presentó en ese momento el cambio de gobierno, lo cuál incluyó al Director de Impuestos Nacionales quien no le ha dado el impulso que necesita el proyecto y con el que contaba anteriormente. Debido a esto el CIS ha efectuado un giro a su estrategia y ha generado un "Piloto Informático" que como su nombre lo indica, probará la bondad de los nuevos sistemas no sólo desde el punto de vista de la eficiencia del "software", sino que la forma de diseñarlo y escribirlo cambiará radicalmente.

A la DIN se le han concedido facultades para que antes de finales de 1.987 efectue una reforma administrativa. Muchos de estos cambios estan incluidos en el Piloto, por lo tanto quizas sea este el punto a través del cual los planteamientos del proyecto se lleven a cabo con mayor celeridad.

C. Piloto Informático

Se ha aprovechado el reemplazo de equipos en que se encuentra el CIS para incorporar al ambiente existente, el nuevo esquema informático desarrollado en el "Proyecto Piloto", el cuál se ha denominado "Ambiente de Entrada de Datos". Este ambiente inicialmente incluye las aplicaciones que tienen que ver con la manipulación de los documentos de renta, ventas y retenedores, así como los pagos efectuados en las regionales que aún no cuentan con teleproceso y todo lo relativo a la recaudación en bancos. Permitirá consultas importantes como los recaudos diarios y diferentes tipos de estadísticas, así como la historia de cualquier documento, se harán simulaciones de manejo operativo y administrativo utilizando una muestra del universo de contribuyentes, demostrando así la oportunidad de la información en el nuevo sistema. Todo ésto utilizando tecnología de bases de datos.

El desarrollo que se haga en este ambiente, tendrá la suficiente flexibilidad para permitir un esquema integrado con los ambientes computacionales actuales involucrando así las aplicaciones existentes en el Piloto Informático.

La experiencia obtenida con el "Ambiente de Entrada de Datos" en el Piloto Informático permitirá que los demás módulos de "software" que no implican manejo físico de documentos, como el recaudo en línea, la fiscalización, la conformación y actualización de la Cuenta Corriente de los Contribuyentes, sean adicionados fácilmente. La coexistencia e integración que existirá con los ambientes actuales va a permitir una continuidad dentro del cambio, debida a la implantación del sistema por etapas.

IV. COMENTARIOS FINALES

Es claro que con el proyecto se logra un sistema eficaz, seguro, ágil, dinámico, barato, oportuno, coherente y sobre todo flexible. Se podrá al fin contar con información que permita analizar y estudiar las distintas situaciones.

Permitirá una forma ágil de cobro y de fiscalización que llevará a una disminución en la evasión y la elusión que se verá reflejada en un **aumento en el recaudo.**

Será un sistema coherente y completo en el cuál las estadísticas y **la contabilidad son un subproducto natural del procesamiento de las declaraciones.**

Como **experiencia en el campo informático** es realmente apreciable la forma como se cambia de un sistema viejo y arraigado en el país a uno totalmente nuevo, con una filosofía y una tecnología totalmente diferentes.

Marca una **nueva era en la forma de escribir "software"** en el Ministerio de Hacienda colocándolo en un sitio de avanzada en el ambiente informático del país.

El principal beneficiado con el cambio **será el estado** porque ademas de contribuir con su modernización, podrá contar con un sistema que satisface las necesidades de la DIN en un tiempo bastante aceptable, que resuelve satisfactoriamente el problema planteado, **seguro** porque es completamente auditable desde cualquier punto, que permite ir variando de acuerdo a las normas tributarias. Indirectamente, la **atención al público mejorará** apreciablemente porque el control al cumplimiento de las obligaciones tributarias será oportuno; la Red Bancaria brindará gran comodidad y rapidez; los archivos de cuenta corriente y la información estarán actualizados a tiempo, y **eficaz** porque las necesidades del contribuyente serán satisfechas en forma ágil y segura.

Los costos de infraestructura bajaran, se estima que solo en equipo se logre un ahorro de más o menos un 35%.

La configuración de una "Unidad Piloto" permitirá dejar vigente un plan de capacitación a través del cuál deberá pasar todo funcionario que desee trabajar en este campo tanto para el CIS como para la DIN.

En el futuro el proceso administrativo que forma parte del proyecto se llevará a una "Unidad Piloto" para lo cuál se escogerá una regional pequeña a partir de la cuál se extenderá el nuevo sistema tributario a todo el país.

Exportable. La modularidad y flexibilidad del sistema, permiten que sea fácil de adaptar a diferentes circunstancias, por lo cuál en otros paises, con organizaciones parecidas a la nuestra, podría prestar un gran servicio.

BIBLIOGRAFIA

Hernàndez C. Alfredo
"Colombia proyecta sus sistema tributario para la dècada del 90"
XX Asamblea General del CIAT,
Abril de 1986, Buenos Aires, Argentina.

Hernàndez C. Alfredo
"Nueva tecnologìa de software para el sistema tributario de Colombia"
Revista Sistemas de la ACIS
Julio de 1986, Bogotà Colombia.

Ministerio de Hacienda y Crèdito Pùblico
Centro de Informaciòn y Sistemas C.I.S.
"Lineamientos generales para la Administraciòn Tributaria de Colombia"
Julio de 1986.

Silvani Carlos
Fondo Monetario Internacional F.M.I.
"Anàlisis y sugerencias para la Administraciòn Tributaria"
Julio de 1986.

Ministerio de Hacienda y Crèdito Pùblico
Centro de Informaciòn y Sistemas C.I.S.
Documento de Soporte Tècnico para la Configuraciòn de los Equipos de la Unidad Piloto
Agosto de 1986.

Bedoya V. Esperanza
Ministerio de Hacienda y Crèdito Pùblico
Centro de Informaciòn y Sistemas C.I.S.
"Descripciòn general del Proyecto Piloto de Impuestos"
Enero de 1987

"SISTEMA DE INFORMACION PARA LA ADUANA NACIONAL DE COLOMBIA".

AUTORES

ALFREDO HERNANDEZ CIFUENTES.
FABIO CAMACHO PARDO.

MINISTERIO DE HACIENDA Y CREDITO PUBLICO

Bogotá D.E., Colombia.

Carrera 8 # 6 - 40.

RESUMEN : El "Sistema de Información para la Aduana Nacional de Colombia" es un amplio plan de sistematización para la Dirección General de Aduanas del Ministerio de Hacienda, enmarcado dentro del programa de modernización, tecnificación y simplificación del Comercio Exterior Colombiano.

El proyecto contempla 6 áreas que son : "Desaduanamiento", "Bodegas", "Jurídico", "Valoración", "Administrativo" y "Correo Electrónico".

La principal área del proyecto es el "Subsistema de Desaduanamiento", encargado de atender los trámites propios de los 7 principales regímenes de la legislación aduanera, en la nacionalización y exportación de mercancías.

Se ha desarrollado un "Prototipo" en el que se implementan las funciones básicas de dicho proceso, sirviendo como herramienta fundamental para el análisis y la especificación del mismo.

Su implementación se realizará en una "Aduana Experimental" como plan a corto plazo, pensando en la ampliación del mismo a 5 ciudades del país en su segunda etapa, y posteriormente, a 14 aduanas regionales para lograr la cobertura nacional del sistema.

1.0 MARCO DE REFERENCIA.

1.1 ENTORNO NACIONAL.

El proyecto del "Sistema de Informaciòn para la Aduana Nacional" se enmarca dentro del ambiente del Comercio Exterior Colombiano, en un esfuerzo estratègico de diversas entidades del gobierno directamente vinculadas al nivel de ejecuciòn de èste.
Los organismos que ejecutan dicha polìtica son principalmente : el Instituto Colombiano de Comercio Exterior, INCOMEX; el Banco de la Repùblica; el Fondo de Promociòn de Exportaciones, PROEXPO; el Departamento Nacional de Estadìsticas, DANE; y la Direcciòn General de Aduanas. Estas entidades se encargan de cumplir y hacer cumplir el règimen de Importaciones, el règimen de Exportaciones, el règimen Aduanero y Arancelario, y el règimen cambiario, base fundamental de nuestra legislaciòn.

El proyecto busca una coordinaciòn operativa e intercambio de informaciòn entre las entidades mencionadas, con el claro objetivo de optimizar y modernizar el funcionamiento del engranaje estatal del comercio exterior Colombiano. Esto aportarà unas herramientas coherentes y oportunas para garantizar el cumplimiento de dicha polìtica y aùn màs, poder planear la elaboraciòn de la misma.

1.2 LA ADUANA NACIONAL.

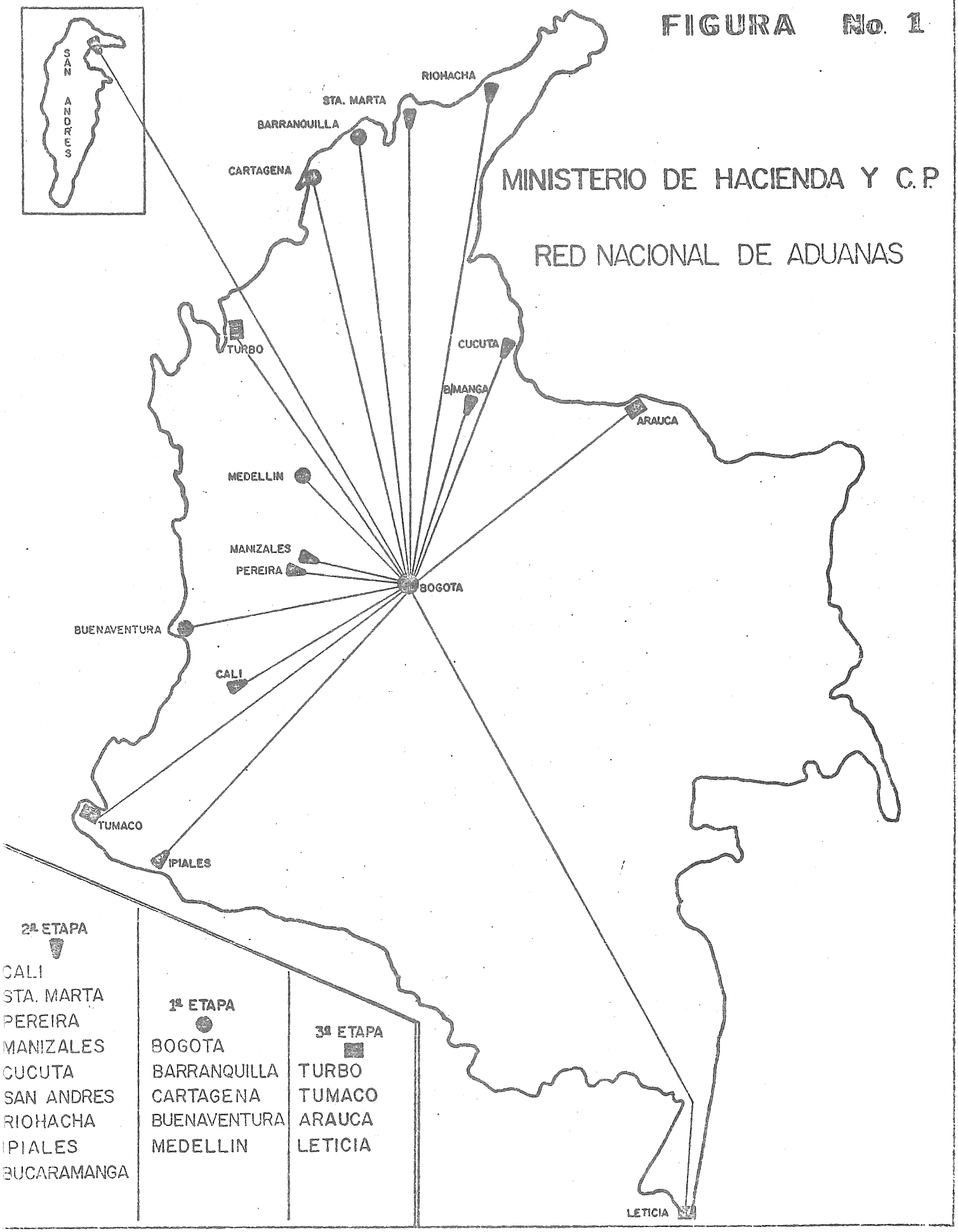
La Direcciòn General de Aduanas del Ministerio de Hacienda y Crèdito Pùblico, es la entidad oficial encargada de controlar la importaciòn, exportaciòn y trànsito de mercancìas que entran y salen del paìs, y ejecutar las actividades que giran alrededor de estos aspectos, como la liquidaciòn y recaudo de impuestos, derechos aduaneros y demàs gravàmenes, la prevenciòn e investigaciòn del fraude al respecto, los mecanismos de control y aprehensiòn de todo tipo de contrabando y la coordinaciòn de actividades con las demàs entidades de comercio exterior.

Finalmente, la Aduana cuenta con 18 Administraciones regionales encargadas de controlar las operaciones de las Importaciones, Exportaciones y trànsito de mercancìas en el territorio Nacional. (Ver Figura No. 1).

1.3 ANTECEDENTES HISTORICOS DE INFORMATICA.

Las diversas entidades del comercio exterior han tenido una heterogènea evoluciòn en la sistematizaciòn de sus procesos, destacando el hecho que el Banco de la Repùblica, el Incomex y el Dane tienen sistemas en funcionamiento hoy en dìa.

FIGURA No. 1


SAN ANDRES
RIOHACHA
STA. MARTA
BARRANQUILLA
CARTAGENA
MINISTERIO DE HACIENDA Y C.P.
RED NACIONAL DE ADUANAS
TURBO
CUCUTA
B/MANGA
ARAUCA
MEDELLIN
MANIZALES
PEREIRA
BOGOTA
BUENAVENTURA
CALI
TUMACO
IPIALES
LETICIA
2ª ETAPA
CALI
STA. MARTA
PEREIRA
MANIZALES
CUCUTA
SAN ANDRES
RIOHACHA
IPIALES
BUCARAMANGA
1ª ETAPA
BOGOTA
BARRANQUILLA
CARTAGENA
BUENAVENTURA
MEDELLIN
3ª ETAPA
TURBO
TUMACO
ARAUCA
LETICIA

Por su parte la Aduana ha realizado varios esfuerzos desde 1.975 para sistematizar parte de su funcionamiento, unos con menor fracaso que otros, teniendo como resultado un sistema de control de manifiestos en Bogotá.

Sin embargo, la Aduana es la más pobre en su desarrollo informático, tanto a nivel de las entidades de comercio exterior, como del Ministerio de Hacienda. Esta situación de desventaja comparativa hizo prioritario replantear su desarrollo y elaborar un plan que cubra esta deficiencia.

2.0 AREAS DEL PROYECTO DEL SISTEMA DE INFORMACION PARA LA ADUANA NACIONAL "S.I.A.N".

El proyecto del S.I.A.N, se incluye dentro del programa bandera de la actual Dirección General de Aduanas, que contempla la reestructuración orgánica y funcional de la misma, así como la modernización, tecnificación y optimización de sus procesos, respaldado en la norma jurídica contemplada en el artículo 102 de la Ley 75 de 1986 mediante la cual se faculta al Gobierno Nacional para dicha modificación.

Con el fin de agilizar los trámites administrativos que deben cumplir los usuarios de la aduana, darles seguridad y transparencia jurídica, controlar el cumplimiento de las obligaciones con el Estado, evitar fraudes, y en una palabra, mejorar la administración aduanera, la Dirección General de Aduanas se ha comprometido en un amplio programa de sistematización, entendiendo por tal la adopción de métodos y herramientas computacionales para el cumplimiento de sus tareas.

El proyecto SIAN se compone de 6 áreas principales que abarcan las prioridades de la entidad, en cuanto al manejo de información. El objetivo es crear una Base de Datos Relacional de Información Aduanera, siendo coherentes con la tecnología y metodología de trabajo que se viene adelantando en el Ministerio.

Estas áreas son:

1.- **Area de Desaduanamiento de Mercancías** : Se refiere a los trámites para la importacion y exportación de mercancías, y dada la importacia del desaduanamiento, se ha decidido que éste sea el que primero se implante en la D.G.A. Esta área será detallada en el punto 3.0.

2.- **Area de Valoración** : Consiste en la creación de la Base de Datos de Valoración de Mercancías, que premitirá establecer métodos y aportar información actualizada sobre los precios de los productos objeto del comercio internacional. Esta Base de Datos servirá como elemento de juicio para el funcionario aduanero que tenga que emitir concepto sobre la materia, permitiendo ésto controlar la subfacturación de productos, incidiendo así directamente en los recaudos de impuestos, además que permite brindar información oportuna al nivel directivo del

gobierno, para el control de la ejecución y para la planeación de la política de comercio exterior del país.

3.- Area de Comunicaciones : El objetivo fundamental es la tecnificación y modernización de los medios de comunicación interna y externa al nivel central de la D.G.A, dadas las características del objeto social de la entidad, garantizando su confiabilidad y seguridad, ya que los actuales recursos son obsoletos, anticuados e insuficientes.

Se plantean dos redes para la comunicación de voz y datos. La primera, ó Red Administrativa, comunicará la Dirección General, las Subdirecciones y demás dependencias del nivel central, y los administradores y comandantes de resguardo a nivel regional; La segunda, ó Red Operativa, se usará para las comunicaciones del personal del Resguardo en todos los distritos aduaneros, siendo integradas las dos redes mediante un sistema PABX. La cobertura de la red para comunicación de datos a nivel nacional se analiza en el punto 3.

4- Area de Bodegas : Servirá como mecanismo básico para la modernización y tecnificación de los procesos de recepción, almacenamiento y salida de mercancías de las bodegas de la Aduana y de aquellas bodegas de terceros debidamente autorizadas, mientras se cumple el proceso de desaduanamiento.

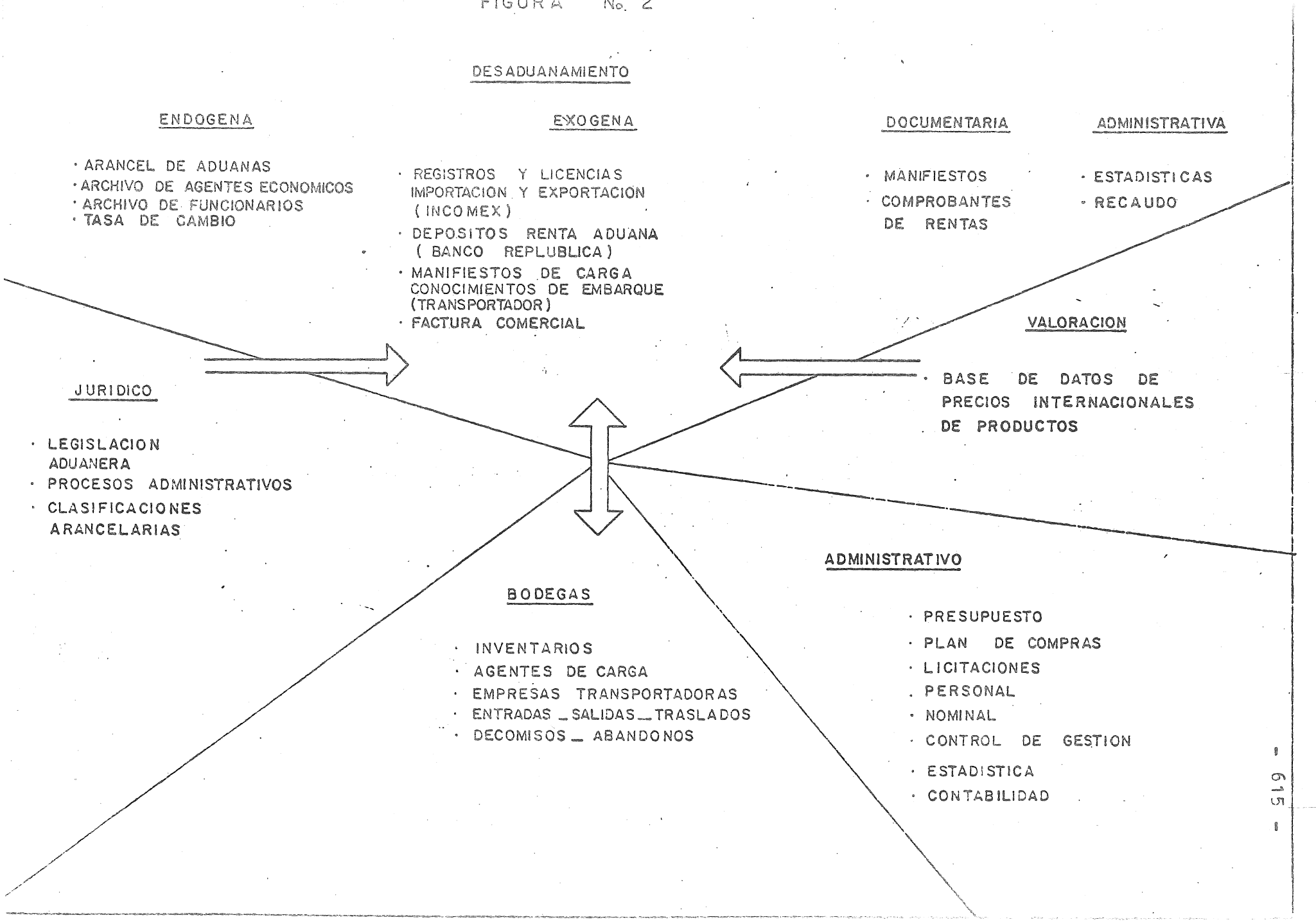
El sistema que se implante deberá permitir la interación entre las áreas de desaduanamiento y bodegas, procesos que están íntimamente relacionados y que son complementarios en toda operación aduanera.

Se busca mejorar funcional y físicamente los procesos, las bodegas, los formularios y formas, y los controles necesarios para el manejo de las mercancías, adoptando herramientas computacionales que permitan esta labor, brindando además la facilidad de obtener información oportuna y altamente confiable, tal como la estimación del inventario en un momento determinado, el cumplimiento de la legislación en lo que se refiere a términos y plazos, acciones legales y administrativas, etc.

5.- Area Administrativa : Incluirá los sistemas automatizados de personal, nóminas, elaboración y control de la ejecución presupuestal, plan de compras, control de inventarios, contabilidad, análisis financiero y estadístico, control de gestión, etc.

6.- Area Jurídica : Buscará la creación de herramientas informáticas para atender las necesidades sobre información legal, jurisprudencial y doctrinaria de la legislación aduanera y arancelaria, así como permitir el establecimiento de controles y seguimientos de los procesos judiciales de interés de la Aduana Nacional, en lo que se refiere a la justicia penal aduanera y los procesos de investigación y vigilancia que adelante la D.G.A.
El desarrollo de éstas 6 áreas se debe realizar permitiendo la interacción entre ellos. (Ver Figura No. 2)

FIGURA No. 2


DESADUANAMIENTO
ENDOGENA
· ARANCEL DE ADUANAS
· ARCHIVO DE AGENTES ECONOMICOS
· ARCHIVO DE FUNCIONARIOS
· TASA DE CAMBIO
EXOGENA
· REGISTROS Y LICENCIAS IMPORTACION Y EXPORTACION (INCOMEX)
· DEPOSITOS RENTA ADUANA (BANCO REPLUBLICA)
· MANIFIESTOS DE CARGA CONOCIMIENTOS DE EMBARQUE (TRANSPORTADOR)
· FACTURA COMERCIAL
DOCUMENTARIA
· MANIFIESTOS
· COMPROBANTES DE RENTAS
ADMINISTRATIVA
· ESTADISTICAS
· RECAUDO
VALORACION
· BASE DE DATOS DE PRECIOS INTERNACIONALES DE PRODUCTOS
JURIDICO
· LEGISLACION ADUANERA
· PROCESOS ADMINISTRATIVOS
· CLASIFICACIONES ARANCELARIAS
BODEGAS
· INVENTARIOS
· AGENTES DE CARGA
· EMPRESAS TRANSPORTADORAS
· ENTRADAS _ SALIDAS _ TRASLADOS
· DECOMISOS _ ABANDONOS
ADMINISTRATIVO
· PRESUPUESTO
· PLAN DE COMPRAS
· LICITACIONES
· PERSONAL
· NOMINAL
· CONTROL DE GESTION
· ESTADISTICA
· CONTABILIDAD

3.0 PROYECTO DE DESADUANAMIENTO DE MERCANCIAS.

3.1. PROCESO DE DESADUANAMIENTO.

Toda transacción de comercio exterior comienza en el Incomex, en donde el interesado presenta una solicitud, bien sea de importación ó exportación, para que sea aprobada por la entidad, según el régimen de importaciones y exportaciones y la política de comercio exterior fijada por el Gobierno. Cuando la transacción se aprueba, se expide el correspondiente "Registro" ó "Licencia".

Cuando se realiza la importación ó exportación física de la mercancía, la Aduana verifica que ésta haya sido de acuerdo a la aprobación del Incomex y bajo la ley. Una vez revisada la mercancía y la declaración (Manifiesto de importación ó de exportación), se pasa a liquidar y verificar la cancelación de los derechos arancelarios y demás impuestos que la operación cause.

Por su parte, el Banco de la República se ocupa del recaudo de los impuestos y derechos aduaneros según la liquidación oficial realizada por la aduana, dejando constancia de su cancelación en el "Depósito Renta Aduana". Una vez realizado el recaudo, se procede a distribuirlo en el Tesoro Nacional, Fondos Comunes y el manejo de divisas, controlando el reintegro de las mismas ó realizando el giro al exterior.

Los trámites de desaduanamiento de mercancías ante la Aduana se reducen a 4 pasos escenciales: Comprobación, Aforo, Liquidación y Caja, convirtiéndose éstas en áreas funcionales en las administraciones regionales.

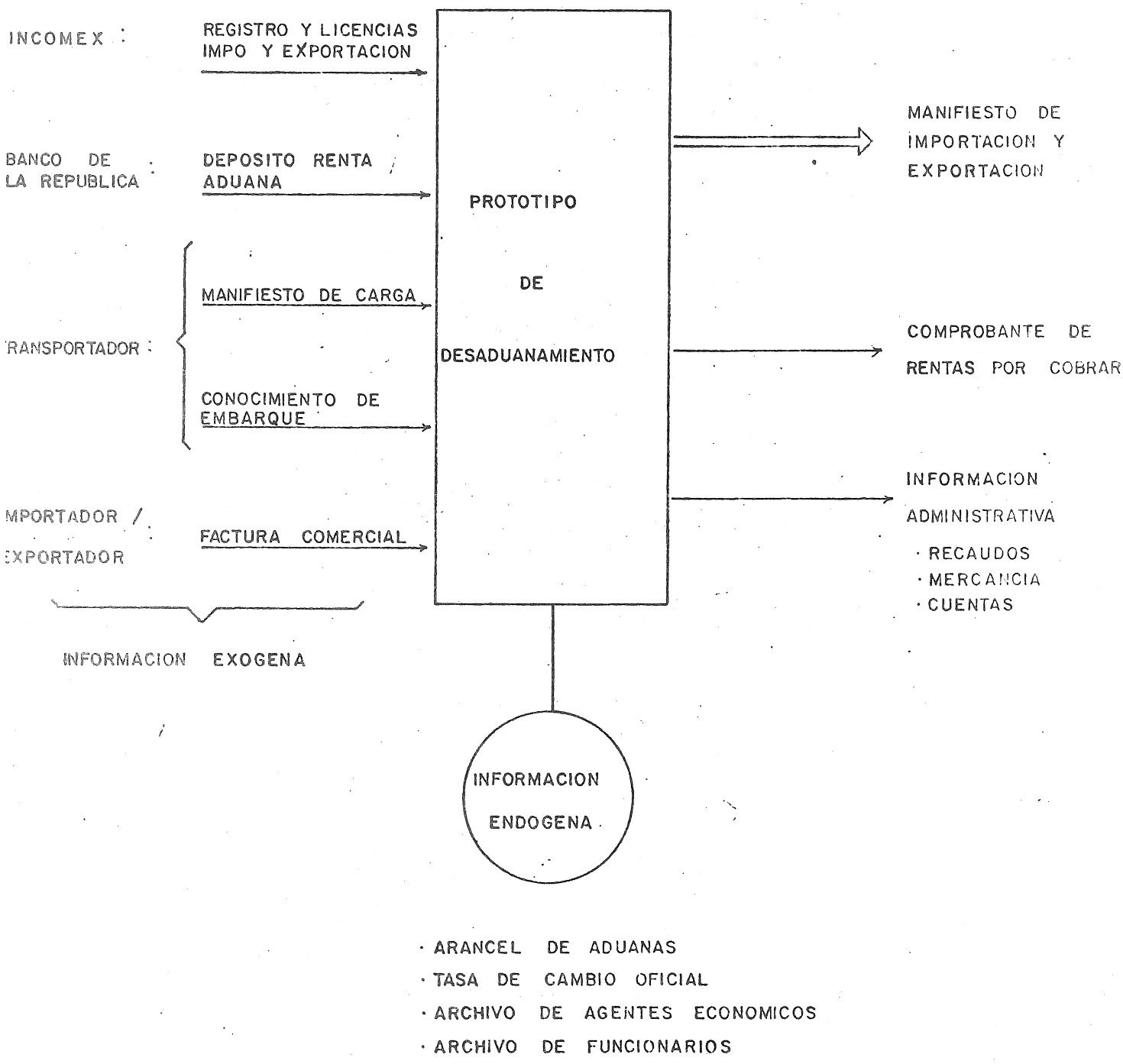
La mercancía objeto de nacionalización es entregada a la Aduana, la que se encarga de hacer la recepción del medio de transporte respectivo y después vigilar la ubicación de la mercancía en las bodegas respectivas.

El Importador, ó su Agente de Aduana, presenta la documentación legal ante la oficina de comprobación. Los documentos que deben presentarse, están relacionados en la Figura No 3.

En síntesis, la oficina de comprobación verifica la legalidad y consistencia de la información contenida en dichos documentos, aceptándola ó rechazándola según sea el caso. Una vez aceptada la documentación, se continúa el proceso de nacionalización por medio del aforo, acto por el cual se realiza la inspección visual de la mercancía.

DISEÑO DEL PROTOTIPO DE DESADUANAMIENTO

FIGURA No. 3



El aforador emite un concepto técnico sobre la descripción, naturaleza, cantidad y valor de la mercancía, ajustándose al arancel de Aduanas de Colombia para fijar las posiciones arancelarias (código de ocho dígitos asignado a cada producto objeto de comercio internacional, según la nomenclatura Nabandina) y el gravámen de cada producto importado.

Con base en esta información, se liquidan los impuestos, los derechos arancelarios, bodegajes, multas y demás conceptos legalmente causados, para que finalmente, el importador cancele dicha suma y pueda retirar su mercancía de las bodegas de la Aduana.

3.2 ANTECEDENTES HISTORICOS.

En la Sección de Finanzas y Sistemas de la D.G.A., se han realizado algunas aplicaciones que atienden parte del funcionamiento de la Aduana. Desde 1983 se tiene en la Aduana Interior de Bogotá un sistema de control de manifiestos de importación, utilizando los recursos de "hardware" y "software" que le proporciona el Centro de Información y Sistemas del Ministerio de Hacienda - C.I.S. (CPU, Mainframe I.B.M. 4381, Discos, Cintas, Terminales, red de teleproceso, etc). Este sistema logró establecer un control parcial de los manifiestos de importación, permitiendo tener información del estado de cuenta del importador en cuanto al pago de derechos e impuestos de aduana, el uso de Registros de importación y la notificación de liquidación oficial, pero no abarca la generalidad del problema en su diseño, ni brinda solución para el mínimo de los procesos de desaduanamiento.

3.3 OBJETIVOS DEL PROYECTO DE DESADUANAMIENTO.

El área de desaduanamiento permitirá atender los procedimientos aduaneros en 7 regímenes ó modalidades principales que contempla la legislación (Decreto 2666 de 1984), en su primera fase :

- Despacho para consumo.
- Importación temporal para reexportación en el mismo estado.
- Importación temporal para perfeccionamiento activo.
- Envíos urgentes e importaciones menores.
- Viajeros y Menajes Domésticos.
- Admisión con franquicia de Derechos.
- Exportación Definitiva.

Se busca implantar un sistema automatizado, que agilice y simplifique el proceso, realice los controles necesarios, brinde información confiable y oportuna y prevea las facilidades para interactuar con las demás entidades que sean necesarias. Tal área debe ser dirigido y controlado por la Dirección General de Aduanas, permitir la interacción con las demás áreas del S.I.A.N. que se implementen en la D.G.A. y ser transportable a las demás aduanas del país.

En cumplimiento de dichos objetivos, se busca específicamente mejorar los procedimientos para la actuación aduanera en este campo, e introducir ayudas computacionales para atender dicha actuación en los regímenes mencionados anteriormente, en una aduana del país escogida por la Dirección General de Aduanas, que será conocida como "Aduana Experimental".

El área de Desaduanamiento deberá :

- Incluir la posibilidad de interacción con las áreas de "valoración", "jurídico", "administrativo", "correo electrónico" y "bodegas" que se implementarán en el futuro en la D.G.A.

- Ser transportable para poder ser aplicado en cualquier aduana del país y ser altamente compatible en diversos equipos de procesamiento.

- Ser parte de un sistema nacional que enlace las administraciones con el nivel central.

- Ser instalable por subprocesos y/o por regímenes, es decir, ser modular, permitiendo en un futuro, la incorporación de los demás regímenes contemplados en la legislación aduanera.

- Permitir la interacción y comunicación computacional con las demás entidades estatales, en especial Incomex y el Banco de la República.

- Agilizar y simplificar en la medida de lo posible los procesos de desaduanamiento de mercancías, dentro del marco legal.

- Incorporar controles efectivos y eficientes para :

  . Verificar y comprobar el cumplimiento de los términos contemplados en la legislación aduanera (plazos de presentación de documentos, de retiro de mercancías, vigencia de garantías, etc.).
  . Examinar las actuaciones de los funcionarios de la Aduana, de los importadores y agentes de aduana que intervienen en el proceso de desaduanamiento.
  . Vigilar los recaudos provenientes de los derechos de importación.
  . Evitar los fraudes

- Unificar los procedimientos fundamentales de desaduanamiento de mercancías para las distintas aduanas del país.

- Brindar información oportuna y confiable de tipo documental, estadística y gerencial.

- Ofrecer seguridad de operación y

- Ser auditable desde el punto de vista administrativo y fiscal.

3.4 ETAPAS DE DESARROLLO DEL PROYECTO.

3.4.1 PROTOTIPO DE DESADUANAMIENTO.

En el segundo semestre de 1985 se realizaron algunas conversaciones interinstitucionales entre el Incomex, el Banco de la Repùblica y el Ministerio de Hacienda, con el ànimo de promover la integraciòn, coordinaciòn de procedimientos y el intercambio de informaciòn para la tecnificaciòn de los procesos pertinentes de cada institución, y asì mejorar el funcionamiento estatal del comercio exterior.

Paralelamente, se realizò una tesis de grado en el departamento de Ingenierìa de Sistemas y Computaciòn de la Universidad de Los Andes, que permitiò hacer un anàlisis teòrico del sistema de desaduanamiento y de los procesos que se realizan actualmente en la Aduana Interior de Bogotà, produciendo una primera aproximaciòn del prototipo de desaduanamiento.

Debido a la falta de una buena especificaciòn del problema de desaduanamiento en el sistema actual, se tomò la metodologìa de construcciòn de prototipos ràpidos que se venìa trabajando en el Ministerio de Hacienda, para entrar a resolver el problema de Aduanas.

Dicha Tesis sirviò de base para que se realizara una posterior ampliaciòn del prototipo, trabajo que se realizò en el Centro de Informaciòn y Sistemas - C.I.S. del Ministerio de Hacienda, utilizando un microcomputador I.B.M. PC/AT y el manejador de bases de datos relacionales dBASE III Plus. Con dichas herramientas se diseñò y programò la primera versiòn del prototipo en septiembre de 1986.

El prototipo es una "Maqueta" computacional en el que se lleva a la pràctica en forma ràpida las acciones que se quieren sistematizar en el ambiente real, permitiendo una mayor interacciòn con las personas directamente relacionadas, para realizar el anàlisis y Especificaciòn completa del problema.

En el prototipo se logran 4 aspectos bàsicos :

- Poder ver ràpidamente còmo funcionarà el sistema completo en el ambiente real y sus principales alcances.

- La integraciòn del flujo de informaciòn interinstitucional con las entidades del comercio exterior colombiano, Incomex, Banco de la Repùblica y Dane.

- La elaboraciòn de los mòdulos que ayudan a cumplir las funciones bàsicas de Comprobaciòn, Aforo, Liquidaciòn, Caja y Archivo, siendo estos mòdulos independientes, compatibles, coherentes y que cubren la parte fundamental del proceso de desaduanamiento.

- La construcción del Manifiesto de Importación (Formulario de declaración de mercancías) a partir de la información externa al Ministerio de Hacienda, ó Exógena, (documentos de las entidades de comercio exterior y de los particulares), y la información propia del Ministerio ó Endógena (Arancel de Aduanas, Archivo de Importadores, Agentes de Aduana y Exportadores, Archivo de los funcionarios aduaneros y la Tasa de Cambio Oficial). Además, se incluye la información de tipo Documentaria para registrar los documentos de aduana en importación y exportación y los comprobantes de rentas, al igual que la producción de información administrativa sobre recaudos según la distribución rentística, la mercancía importada y exportada, etc. (Ver Figura No. 3).

El prototipo fué evaluado por directivos y funcionarios de la Aduana, por representantes de las entidades estatales de comercio exterior, el sector privado y ususarios de la Aduana, y los proveedores de "hardware" y "software" en el país, siendo de valiosa ayuda los permanentes aportes de cada sector para la definición del problema, hasta producir la versión número tres en febrero de 1987.

3.4.2. ESTRATEGIA DE DESARROLLO A CORTO PLAZO : ADUANA EXPERIMENTAL.

a. GENERALIDADES : A corto plazo se tiene prevista la terminación de la cuarta versión del prototipo, en la que se introducen los procedimientos de los siete regímenes mencionados, se modifica la actuación de aforo, se fusionan las labores de liquidación y caja y se contemplan algunos otros aspectos mencionados por las diversas personas que han interactuado con las versiones anteriores.

El área de desaduanamiento de mercancías será la primera en implantarse en la Aduana Nacional, dada su importancia. Con el fin de garantizar el éxito de esta labor, corregir oportunamente las fallas y tener los menores riesgos, esta área se implantará en una Administración de Aduanas determinada, la cual se conocerá como "Aduana Experimental", de cuyos resultados se extraerán las condiciones y recomendaciones técnicas y administrativas para la implantación de este área y los demás a nivel nacional.

El montaje de la "Aduana Experimental" será financiado por el Programa de las Naciones Unidas para el Desarrollo - P.N.U.D., y el Banco Mundial, concretamente en la adquisición de "hardware" y "software" de base, y en la contratación de servicios técnicos para el desarrollo del "software" necesario.

b. ETAPAS DEL PROYECTO : Para cumplir con los objetivos planteados en el punto 3.3. y realizar el montaje de la Aduana Experimental, se deberán desarrollar las etapas de :

- Análisis Corto.
- Diseño.
- Programación.
- Generación de las "Bases de Datos de Desaduanamiento".
- Pruebas, Capacitación y Montaje de la Aduana experimental.
- Ajustes y Afinamiento.
- Evaluación, transferencia tecnológica y Recomendaciones.

Lo anterior alrededor del proceso de desaduanamiento y sus operaciones básicas (comprobación, aforo, liquidación, auditoría, caja y archivo), partiendo del prototipo.

Se realizará un análisis corto de los procedimientos, flujos de información y formularios utilizados en el proceso. Posteriormente se afinarán y/o diseñarán de acuerdo a las condiciones técnicas, administrativas y legales. Se adelantará el diseño lógico del "software" de cada operación básica y de la Base de Datos de Desaduanamiento, siguiendo la tecnología "Relacional" que se ha trabajado en el Ministerio , para producir el modelo Entidad - Relación, incluyendo la información Endógena, Exógena, Administrativa y de seguimiento del sistema, según el prototipo. El diseño físico de la misma se deberá hacer teniendo en cuenta la compatibilidad con la información del Incomex, el Banco de la República y del Dane. Se escribirán los programas que atiendan los pasos fundamentales con los controles respectivos.

Se pondrá en marcha el plan de capacitación para los usuarios del sistema y para los demás funcionarios de la Aduana Experimental, al tiempo que se realizarán las pruebas operativas y funcionales, tanto de la información interinstitucional, como del "software" de cada proceso, llevando a cabo los ajustes que el "software", el "hardware" y los procedimientos administrativos requieran. Finalmente, se evaluará el sistema implantado con la participación de los funcionarios y usuarios de la aduana y las demás entidades del gobierno, sobre los resultados obtenidos y el impacto en la atención al público.

En una última etapa de recomendaciones, se deberán recopilar y analizar las experiencias logradas en el montaje de la aduana experimental, generando un documento donde se plasmen las recomendaciones para la implantación del Sistema de Desaduanamiento de mercancías a nivel nacional, de manera que puedan ser evaluadas por la Dirección General de Aduanas para el plan de acción del montaje nacional.

c. DESCRIPCION DEL "HARDWARE" Y "SOFTWARE" PARA LA ADUANA EXPERIMENTAL.

Factores que se tuvieron en cuenta para la configuración :

La atención de estos regímenes se realiza en diversos lugares geográficos dentro de la ciudad, como es el Ministerio de Hacienda y Crédito Público, la Aduana Interior de Bogotá D.E., el

Aeropuerto El Dorado y las bodegas de la aduana en el barrio Alamos. Ademàs, el proyecto contempla la comunicaciòn con los computadores del Incomex, el Banco de la Repùblica y el Dane.

La Aduana Experimental, en este caso la Aduana Interior de Bogotà D.E., servirà como primer paso para la sistematizaciòn de la Aduana Nacional en cuanto se refiere al desaduanamiento de mercancías. El montaje nacional de este subsistema incluirá la adquisiciòn de otros equipos de computaciòn para las principales aduanas del país, y adicionalmente, un esquema Distribuido de las principales Bases de Datos.

La elaboraciòn de la configuraciòn planteada se realizò con base en el prototipo, que permite especificar el problema y obtener un diseño aproximado de còmo se implementaràn las funciones bàsicas de desaduanamiento que ya se mencionaron. Se realizò un estudio de tiempos actuales de las funciones bàsicas de cada oficina, y del tiempo esperado con la utilizaciòn del sistema; del nùmero de funcionarios directamente relacionados con la operaciòn y una estimaciòn de las carga de trabajo de cada funcionario que utilizarà el sistema de desaduanamiento. Por otra parte, el volùmen de documentos (ò de operaciones de importaciòn y/o exportaciòn) tramitados en la Aduana de Bogotà D.E., incluyendo los 7 regímenes mencionados, alcanza un promedio de 120.000 al año.

Con base en èste prototipo y en los datos anteriores, se llegò a la estimaciòn de las necesidades de pantallas e impresoras, el espacio en disco y, en general, los recursos de computaciòn tanto de "hardware" como de "software" requeridos.

La metodología que se ha seguido en el Ministerio de Hacienda es la de realizar el análisis y el diseño del sistema para luego lograr una configuraciòn òptima. Dadas las necesidades del momento, se ha tenido que estimar dicha configuraciòn, basados en los estudios ya mencionados. El prototipo ha sido fundamental para esta tarea, permitiendo un considerable ahorro de tiempo en el proceso de contrataciòn de dichos requerimientos. Los componentes de dicha configuraciòn son :

"HARDWARE".

- Un superminicomputador ò Unidad Central de Proceso (UCP), con Memoria principal mínima de 8 megabytes, Bus de datos de al menos 32 bits.

- Unidades de discos fijos y sus controladores. Se estima que la base de datos de desaduanamiento, los recursos necesarios para el sistema operacional, el manejador de bases de datos, los indices, los programas y demàs espacio auxiliar ocuparàn un total de 800 Megabytes de almacenamiento en discos fijos ya formateados.

- Se requieren unidades de "Back-up" y sus controladores para realizar copias de archivos y para el intercambio de informaciòn magnètica interinstitucional.

- Pantallas y sus multiplexores : Se requerirán 36 unidades. Las pantallas serán distribuidas inicialmente de la siguiente manera

remotas :

- Aduana Interior de Bogotá	20 Unidades.
- Aeropuerto El Dorado.	7 Unidades.
- Bodega de Alamos.	2 Unidades.

locales :

- Ministerio de Hacienda.	7 Unidades.

- Impresoras : Se necesitará una Impresora Rápida y catorce (14) Impresoras de menor velocidad, que se conectarán para trabajar junto a las pantallas de los lugares remotos y el nivel central de la D.G.A.

COMUNICACIONES

- Se tendrán las unidades de control y/o proesasores de comunicaciones requeridos para la configuración de la Aduana Experimental, es decir, para garantizar la comunicación entre la D.G.A, la Aduana Interior de Bogotá, las bodegas de los Alamos, el aeropuerto El Dorado y las demás entidades mencionadas. Además se tendrán Modems sincrónicos, para la transmisión de archivos al computador del Incomex, el Banco de la República y el Dane, para transmisión de datos preferiblemente a velocidades entre los 2400 y 9600 Baudios en cada unidad.

"SOFTWARE" :

El "software" que se utilizará está enmarcado dentro de las políticas informáticas del Ministerio, principalmente en el uso de tecnología que permita su portabilidad y el uso de Bases de Datos Relacionales. El "software" de que se dispondrá es el siguiente :

- Sistema Operacional : UNIX (preferiblemente compatible con el sistema V.3 de AT & T) ó "UNIX-LIKE" COMPLETO .

- Base de Datos Relacional : Se espera el paquete completo, el cual usualmente se compone del manejador de Bases de Datos (MBD) con diccionario de datos, generador de reportes, generador de formas de pantalla, lenguaje de interrogación, generador de aplicaciones, paquete de comunicaciones e interfaz con lenguajes de alto nivel e incorporar facilidades para el manejo de bases de datos distribuídas. Deberá además contar con facilidades para transferencia de archivos particularmente a los computadores de INCOMEX, DANE y el Banco de la República, y posteriormente a los de otras administraciones de aduanas del país.

Se tendrá un "software" utilitario, tal como Sort/Merge, lenguaje de programación "C", "software" para comunicación con Microcomputadores IBM-XT / IBM-AT o compatibles para transferencia de archivos y para la emulación de pantallas locales, para comunicación con computadores vía "MODEM" bajo

diferentes protocolos, para comunicaciòn con redes locales bajo protocolos preferiblemente TCP/IP(IEEE 802.3), correo electrònico y para conexiòn con computadores del ambiente IBM's 43XX que son los que se encuentran en el C.I.S.

Finalmente, algùn "software" utilitario adicional, por ejemplo : Paquete de gràficos, hoja electrònica, Procesador de palabra, Paquete estadìstico, "software" para administraciòn de cintas magnèticas, de auditorìa y Utilidades para el monitoreo y reporte del desempeño del computador, la base de datos, la red, etc.

AMBIENTE DE DESARROLLO.

El ambiente de desarrollo debe estar constituido por los equipos ("Hardware"), programas ("Software") y la red de comunicaciones necesarios para el desarrollo del anàlisis, diseño, programaciòn, prueba, documentaciòn y puesta en marcha de las aplicaciones que trabajarán ò funcionarán en el ambiente de producciòn descrito anteriormente.

El "hardware" para el ambiente de desarrollo està orientado a tener estaciones de trabajo ("Workstations"), con determinadas caracterìsticas de compatibilidad con el computador, y algunas impresoras de tipo laser.

En cuanto al "software" para el ambiente de desarrollo, interesa tener un sistema operacional y un manejador de Bases de Datos Relacionales compatibles con los anteriores, y utilitarios para anàlisis, diseño, desarrollo, prueba, documentaciòn y puesta en marcha de proyectos de desarrollo de "software".

En el ambiente de desarrollo se producirà el software del subsistema de desaduanamiento, para luego exportarlo al ambiente de producciòn descrito anteriormente, haciendo indispensable la premisa de la portabilidad del mismo.

3.4.3. ESTRATEGIA DE DESARROLLO A LARGO PLAZO : MONTAJE NACIONAL.

Con base en la experiencia de la Aduana Experimental, se ha pensado ampliar la cobertura del àrea de desaduanamiento a las demàs aduanas del paìs, de manera que se tenga una cobertura nacional y se establezca una comunicaciòn entre el nivel central y las regionales.

Dicha ampliaciòn se realizarà en tres etapas, segùn la prioridad de las administraciones regionales respectivas.

La primera etapa consistirà en la adaptaciòn del "software" de dicha àrea para poder ser aplicado en cuatro aduanas màs, para conformar una red de cinco (5) ciudades en total. Se ha pensado que sean las de mayor importancia del paìs y que por tanto representan el mayor volùmen de transacciones y recaudo.

En esta etapa se ha previsto la adquisición de computadores independientes para dichas aduanas, de manera que atiendan las necesidades de dicha regional y eventualmente puedan expandirse para cubrir las de otras administraciones más pequeñas. Se prevee además un esquema distribuido para las bases de datos que contengan información propia de cada aduana, manteniendo centralizada la información endógena y los consolidados nacionales. Se ha pensado que el procesamiento sea descentralizado, manteniendo interconectados los diferentes computadores con el nivel central.

En la segunda etapa se ampliará la red de comunicaciones y la infraestructura computacional para atender el desaduanamiento en un total de 14 ciudades del país. Esta estrategia deja ver que existen 4 aduanas que son demasiado pequeñas, en las que el volúmen de importaciones y exportaciones es bastante bajo y el recaudo no es significativo. Estas aduanas seguramente serán Tumaco, Turbo, Arauca y Leticia, que conforman la tercera etapa del proyecto. (Ver Figura No. 1).

4.0 COMENTARIOS FINALES.

Los comentarios finales están dentro de dos líneas que enmarcan la solución del S.I.A.N. : la estrategia organizacional y la estrategia informática.

El cambio de administración del Gobierno Nacional, ha permitido dar impulso a una reforma institucional, buscando con esto la reestructuración orgánica y funcional de la Aduana y la modernización, tecnificación y simplificación administrativa de la misma. Esto se ve resaltado por el total compromiso y respaldo del nivel superior de la Dirección General de Aduanas y el Ministerio de Hacienda, que ha participado en la planeación y definición del alcance del proyecto y sus áreas, en los aspectos administrativos y en la interacción con el prototipo de desaduanamiento para llegar a su cuarta versión. Este compromiso permitirá resolver, en parte, la resistencia al cambio, por parte de los funcionarios de la aduana.

La estrategia organizacional incluye también el aspecto demasiado importante de la integración, coordinación e intercambio de información interinstitucional con las demás entidades del comercio exterior colombiano, como son el Incomex, el Banco de la República y el Dane. Este convenio entre las diferentes entidades permitirá completar coherentemente el ciclo de una transacción de comercio internacional desde el punto de vista de las entidades estatales participantes, obtener información certera que permita controlar el cumplimiento y realizar la planeación de la política de comercio exterior que fije el gobierno.

Se ha creado un comitè interinstitucional asesor de los proyectos de informàtica para la Direcciòn General de Aduanas, cuyo objetivo es el de lograr la integraciòn de las entidades mencionadas, garantizar la continuidad del proyecto, y en general el desarrollo informàtico del comercio exterior del paìs, en el que tienen participaciòn, entre otros, altos directivos y Gerentes de Sistemas de entidades del gobierno como el Banco de la Repùblica, el Incomex, el Banco Central Hipotecario, Planeaciòn Nacional y el Director del Dane, quien a su vez preside la Comisiòn Nacional de Sistemas.

Por ùltimo, cabe resaltar la participaciòn que se ha logrado de los diferentes gremios usuarios de la Aduana, que permiten percibir el punto de vista de ellos y sus necesidades reales que deben ser tenidos en cuenta para el desarrollo del proyecto.

Desde el punto de vista informàtico, ha sido fundamental la metodologìa de prototipos en el desarrollo del àrea de desaduanamiento. Esta metodologìa que se ha adoptado en el Ministerio de Hacienda, ha dejado ver sus bondades en el marco de la Aduana, en el que los principales logros han sido :

- Evolucionar en diferentes versiones refinadas del prototipo de desaduanmiento, para lograr una mejor Especificaciòn del problema.
- Vincular a los diferentes usuarios ya mencionados y hacerlos partìcipes activos del proyecto.
- Economizar en costos en el desarrollo del proyecto, ya que se considera que una correcta especificaciòn reducirà los problemas en las posteriores etapas del mismo.
- Realizar un diseño lògico del "software" y la Base de datos, permitien èsto llegar a una estimaciòn de la complejidad del desarrollo del proyecto real y de la configuraciòn del "hardware" y "software" que se utilizarà en el mismo.
- Ser coherente con la polìtica de desarrollo de proyectyos de "software" en el Ministerio de Hacienda, promoviendo el uso de herramientas para el desarrolo de prototipos en las diferentes dependencias.
- Definir las ventajas de las nuevas tecnologìas de "hardware" y "software" para la Aduana Nacional, en lo que se refiere a soluciones descentralizadas y no dependientes del C.I.S., al uso cada vez con mayor èxito de las Bases de Datos Relacionales, de sistemas operacionales como Unix ò Unix-like, y de la aplicaciòn de los microcomputadores a diversas àreas funcionales para desarrollar sus propias aplicaciones mediante el uso de hojas electrònicas, procesadores de texto y manejadores de bases de datos relacionales.
- Definir como prioridad el patrimonio del "Software" para la Direcciòn General de Aduanas, teniendo como premisa bàsica el criterio de Portabilidad del mismo, pensando en el objetivo final de tener una cobertura nacional de las 6 àreas descritas.
- Crear la conciencia en los directivos de la entidad, de la importancia que debe tener el àrea de Sistemas, de manera que se ubique coherentemente en la organizaciòn, segùn la importancia de los servicios que presta.

B I B L I O G R A F I A

- Decreto 2666 de 1984.
"Por el cual se revisa parcialmente la legislación aduanera."
República de Colombia.

- Camacho, Pardo Fabio.
"Sistema de Información para la Aduana Nacional."
Tesis de grado.
Departamento de Sistemas y Computación.
Facultad de Ingeniería.
Universidad de Los Andes.
Agosto de 1986.

- Ministerio de Hacienda y Crédito Público.
Hernandez Cifuentes, Alfredo.
Camacho Pardo, Fabio.
"Sistema de Información Para la Aduana Nacional".
Octubre de 1986.

- Ministerio de Hacienda y Crédito Público.
Centro de Información y Sistemas - C.I.S.
Dirección General de Aduanas.
"Red de Comunicaciones de Enlace Nacional Para la Dirección General de Aduanas".
Abril de 1987.

- Ministerio de Hacienda y Crédito Público.
Hernandez Cifuentes, Alfredo.
Camacho Pardo, Fabio.
"Términos de Referencia Para el Diseño y la Implantación del Proceso Automatizado de Desaduanamiento de Mercancías en una Aduana Experimental".
Julio de 1987.

- Ministerio de Hacienda y Crédito Público.
Hernandez Cifuentes, Alfredo.
Camacho Pardo, Fabio.
"Documento de Soporte Técnico Para la Configuración de los Equipos de una Aduana Experimental".
Julio de 1987.

- Ministerio de Hacienda y Crédito Público.
Hernandez Cifuentes, Alfredo.
Camacho Pardo, Fabio.
"Términos de Referencia Para el Diseño y la Implantación del Proceso Automatizado de Manejo de Bodegas en una Aduana Experimental".
Julio de 1987.

LA SISTEMATIZACION DE LA SUPERINTENDENCIA BANCARIA COLOMBIANA

AUTORES : Alfredo Hernández Cifuentes
Ministerio de Hacienda y Crédito Público
Carrera 8a No 6 - 40, Bogotá

Rodrigo Nassar Montoya
Superintendencia Bancaria
Carrera 7 No 7-56, Bogotá

La Superintendencia Bancaria de Colombia, organismo adscrito al Ministerio de Hacienda y Crédito Público encargado de velar por el adecuado comportamiento del Sistema Finaciero, ha venido desarrollando su proyecto de Sistematización desde el año de 1983, tratando de enfrentar el atraso tecnológico que sufre respecto a las entidades que vigila. El proyecto procura solucionar a mediano y largo plazo los problemas en manejo de información y utiliza la tecnologia relacional y los conceptos de "software" portable para su desarrollo. El articulo describe los antecedentes, las politicas seguidas, las bases de datos a ser implantadas, la infraestructura tecnológica utilizada, y el estado actual del proyecto asi como su proyección hacia el futuro.

INTRODUCCION

La Superintendencia Bancaria de Colombia según el decreto 1939 de Junio de 1986, es un organismo de caracter técnico adscrito al Ministerio de Hacienda y Crédito Público que tiene como principales objetivos asegurar la confianza Pùblica en el sistema financiero y velar porque las instituciones que lo integran mantengan permanente solidez económica para atender sus obligaciones.

Es también competente a la Superintendencia Bancaria evitar que las personas no autorizadas conforme la ley ejerzan actividades financieras, hacer cumplir las normas que rigen el sector financiero atendiendo las prelaciones que trace el Gobierno Nacional para el manejo de la política monetaria, crediticia, financiera y de cambio exterior, así como adoptar políticas de inspección y vigilancia dirigidas a acogerse a nuevas prácticas Bancarias y desarrollos tecnológicos.

Entre las acciones para desarrollar los objetivos y funciones antes señalados, se cuentan la autorización, creación, fusión, transformación, liquidación de las instituciones bajo su control, así como la apertura, traslado y cierre de sucursales y agencias establecidas en territorio nacional o en el exterior.

Dentro de los diversos tipos de información que alimentan la entidad podemos citar cifras económicas, estados financieros, antecedentes de directivos del sector financiero, conceptos de peritos, planes de seguros de capitalización, respuestas a interrogatorios y quejas del público. Después de ser tramitada y procesada ésta información, se producen indicadores financieros, informes de inspección, disposiciones, órdenes, sanciones, y multas entre otros.

La entidad cuenta hasta el momento con cerca de 900 empleados, que colaboran en su gran mayoria con la labor de vigilancia e inspección desarrollada, a través de tres delegaturas que reportan al Superintendente Bancario las diferentes novedades de cada uno de los sectores del sistema financiero Colombiano.

Los decretos 1939 ,1940 y 1941 del 19 de Junio de 1986, modifican la estructura organizacional y el régimen de contratación para darle a la entidad una estructura moderna que facilite y mejore la labor de vigilancia e inspección realizada.

Este articulo describe el proyecto de sistematización de la Superintendencia Bancaria, sus antecedentes, las políticas seguidas durante su ejecución, las bases de datos objeto del proyecto, así como la infraestructura tecnológica utilizada durante su desarrollo y el estado actual y proyección hacia el futuro del mismo.

ANTECEDENTES

Los primeros avances en sistematización se iniciaron a mediados de la década de los 70 con ayuda de los equipos del DANE donde se procesaba la nómina y algunos informes de tipo estadistico. Dichas aplicaciones fueron trasladadas a los equipos del Ministerio de Hacienda y Crédito Público a mediados de la década de los 80. Adicionalmente existian esfuerzos aislados en sistematización en áreas como la de Vivienda, Registro e Intermediarios de Seguros.

El proyecto se inició a finales de 1983 con la compilación de la información que la entidad manejaba, con la intención de agrupar los diferentes esfuerzos y entender el problema de información de una manera global. Se definieron dentro de este proceso los diferentes subsistemas, asi como un plan de prioridades de desarrollo y estimación de recursos. Esta primera fase finalizó en Septiembre de 1984.

A través de un contrato de asesoria externa, se procedió inmediatamente con el diseño lógico o conceptual de los subsistemas identificados como prioritarios, y una vez revisados y aprobados los diseños por parte de la Superbancaria, se iniciaron los diseños Físicos, presentados en Noviembre de 1985 y solo recibidos en Abril de 1986, luego de varios ajustes.

Desde Octubre de 1985, el Ministerio de Hacienda y Crédito Público asumió la dirección técnica del proyecto, y en coordinación con las directivas de la entidad, conformó un grupo de asesores que ha venido trabajando en la revisión de los diseños de los subsistemas (los cuales fueron reorientados hacia la tecnología de bases de datos relacionales), adquisición de equipo, elaboración de prototipos rápidos e implantación de las bases de datos principalmente.

POLITICAS

El proyecto se enmarcó dentro de la política general de decentralización de la informática del Ministerio, buscando que la entidad asumiese la responsabilidad sobre sus aplicaciones en esta área.

La sistematización se ha venido desarrollando en paralelo con otros proyectos de suma importancia para la Superintendencia como son la reestructuración orgánica que incorpora una División de Sistemas a la planta de personal, que además de recoger los grupos o individuos aislados que trabajaban en informática, es necesaria para ofrecer soporte técnico, mantener y desarrollar aplicaciones y colocar en producción las diferentes bases de datos de la entidad.

Sabiendo que la Superintendencia recibe gran cantidad de datos

que al momento no es posible procesar, se están estudiando diversas formas para reducirlos. Dentro de la reestructuración administrativa se incluyó la Oficina de Control interno la cual entre otros estará a cargo de racionalizar y canalizar la información que se solicita.

La entidad estará en capacidad de dar diversos tipos de informes, los cuales por su oportunidad podrán ser utilizados como herramienta gerencial en la toma de decisiones por las instituciones y público en general.

El criterio principal en la selección de tecnología se fundamentó en la necesidad de suministrarle a la entidad sistemas a largo plazo, quizas de vida útil no menor de 20 años, lo cual se piensa lograr con "software" trasportable a nivel de sistema operacional (UNIX) y a nivel de aplicaciones (manejador de bases de datos relacional ORACLE).

También se encontró interesante dentro del contexto de la Superintendencia la utilización de los microcomputadores como herramienta complementaria para el análisis financiero, desarrollo de aplicaciones muy particulares en los diferentes grupos funcionales y sistematización de áreas como la administrativa, esto con la utilización de hojas de cálculo, bases de datos y paquetes especiales de nómina, almacén, personal y presupuesto por ejemplo.

BASES DE DATOS DE LA SUPERBANCARIA

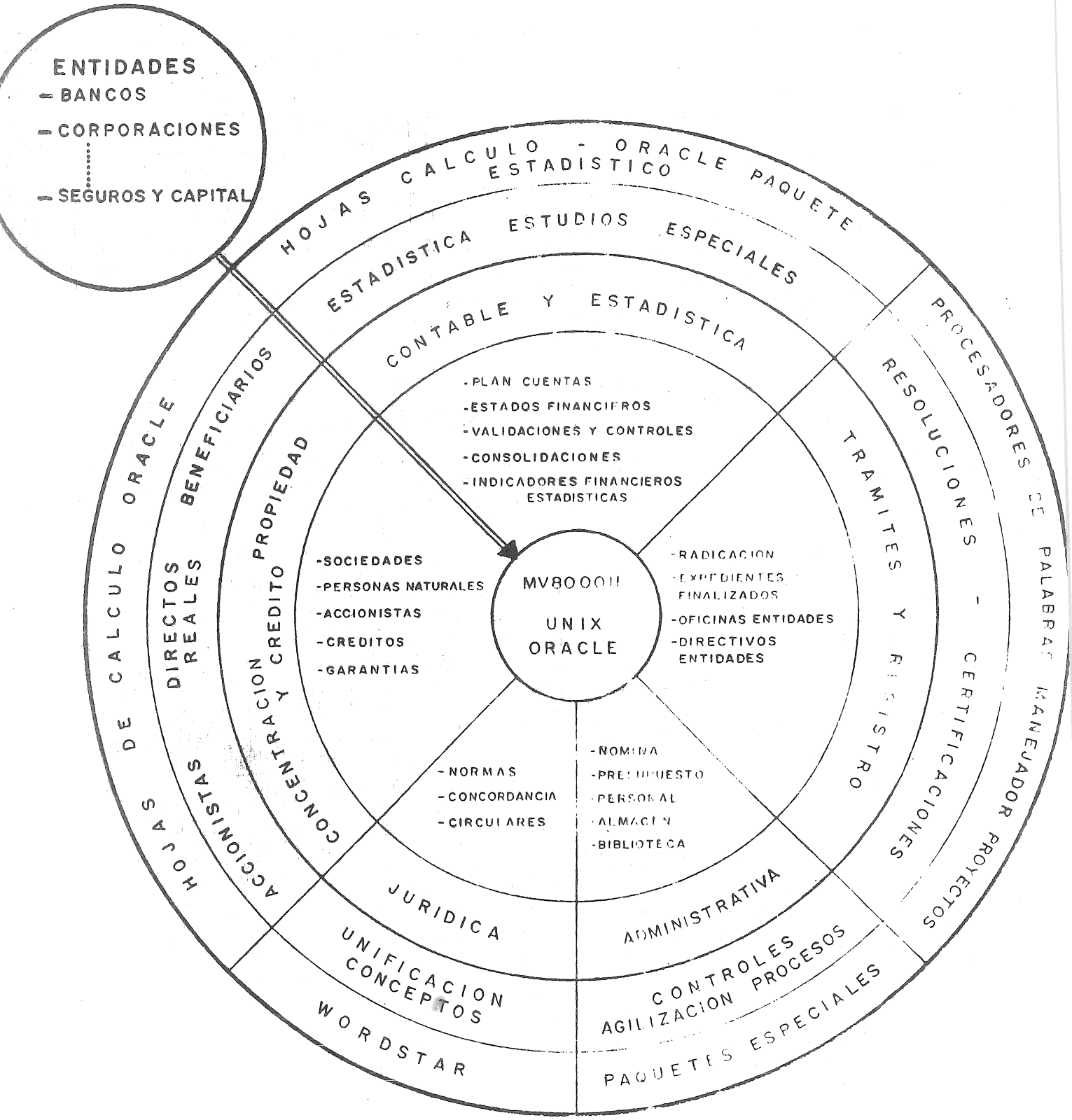
Partiendo de la definición de los subsistemas, con la nueva orientación técnica que le dió el Ministerio de Hacienda al proyecto, se inició el diseño de bases de datos utilizando el modelo relacional.

En el centro del gráfico adjunto se muestra el "software" base de la instalación y en los anillos periféricos de adentro hacia afuera, por aplicación : componentes básicos, nombre o designación, principales resultados y paquetes utilizados en los micros como complemento y/o soporte. Adicionalmente, se muestra la interfaz del sistema con los diferentes tipos de entidades.

A continuación, describimos brevemente cada una de las bases de datos de mayor interés por su importancia dentro de la entidad y por su peculiaridad debido a la naturaleza de la Superintendencia:

BASE DE DATOS DE CONTABLE Y ESTADISTICA

La Superintendencia recibe aproximádamente 8'000.000 de cifras contables anualmente de cerca de 330 entidades del sector financiero: Bancos, Corporaciones Financieras y de Ahorro y Vivienda, Compañias de



BASES DE DATOS DE LA SUPERINTENDENCIA BANCARIA

Financiamiento Comercial, Almacenes Generales de Depósito, Compañias de Seguros y Capitalización, Intermediarios de Seguros entre otras.

Se reciben informes semanales, balances consolidados (mensual), balances descompuestos (trimestral), informes de agencias (mensual y trimestral).

Alrededor de 60 funcionarios están dedicados casi exclusivamente a la verificación aritmética de cifras, sin tener ni el tiempo ni los recursos para analizar el verdadero estado de la entidad que vigilan a través del tiempo, o con respecto a otras entidades de su grupo o a otras entidades del sector financiero.

Actualmente se adelanta la elaboración de un plan único de cuentas para el sector financiero, el cual permitirá unificar la forma en que se solicita y recibe la información, así como los criterios de control para los diversos tipos de entidades, incorporando nuevas técnicas y metodologías de análisis. Adicionalmente se pretende eliminar la información que se solicita y no se procesa.

La base de datos de Contable y Estadística permitirá:

- Recibir gran porcentaje de la información vía MODEM o medios magnéticos. Es decir, las entidades vigiladas podrán conectar sus computadores al computador de la Superintendencia para enviar sus informes financieros o eventualmente podrán enviar la información en cintas magnéticas o "diskettes". En los casos en que no sea posible utilizar alguna de las modalidades anteriores, como último recurso, se recibirán en formatos de papel para ser digitados en la Superintendencia.

- Validar aritméticamente los datos recibidos (suma subcuentas igual a suma cuenta, etc.) en forma automática, reportando a los analistas las diferentes inconsistencias. En una segunda fase se entregará el "software" o su especificación para que las entidades envíen la información validada.

- Utilizar las cifras contables validadas para procesar los diferentes controles exigidos por ley a las entidades vigiladas (verificar montos en inversiones, pasivo para con el público, encaje etc.).

- Verificar el estado de la entidad a través de indicadores financieros que identifiquen aquellas instituciones que tienen comportamientos aceptables o por el contrario "prender luces rojas " sobre aquellas que deben ser vigiladas de forma especial.

- Elaborar consolidaciones por diferentes conceptos que permitan

agrupar por ejemplo las entidades por tipos (Bancos, Corporaciones de Ahorro y Vivienda etc.), diferentes tipos de entidades por regiones y por municipios.

- Generar la facilidad de transmitir la información validada a dependencias como Análisis de Balances y Estadística para que en los microcomputadores se elaboren estudios e informes especiales en hojas electrónicas o bases de datos particulares.

- Facilitar la consulta a través de terminales localizadas en las diferentes dependencias de la información Contable y en general sobre los controles, consolidaciones, estadísticas.

Gran parte del éxito de este subsistema se concentra en los resultados del plan único de cuentas, la reducción de la información que este aporte, la aceptación y acomodación de las entidades a la nueva metodología, y la asimilación de la Superbancaria, especialmente de la Dirección de Estadística e Informática, a la nueva estructura de análisis.

Se espera que el oportuno procesamiento de la información, además de permitir el análisis preventivo que substituya el correctivo, servirá de herramienta gerencial no solo a las directivas de la Superintendencia y Gobierno en general, sino al sector privado, procurando que el organismo de vigilancia no solo sea receptor de datos sino que adicionalmente suministre las estadísticas y resúmenes oportunos para la toma de decisiones del público en general.

BASE DE DATOS DE TRAMITES Y REGISTRO

En promedio se manejan en la Superintendencia cerca de 500 trámites por día . Estos se generan tanto externa como internamente y actualmente la información de trámites y registros se encuentra dispersa por las diferentes dependencias dificultando el control generalizado de los mismos.

El registro de entidades, sucursales, directivos, accionistas y estatutos se desarrolla de una forma manual ofreciendo pocas posibilidades de consulta oportuna a la información.

El proyecto pretende manejar el trámite desde que se inicia, ya sea en radicación(externo) o en las diferentes dependencias (interno), siguiendo su ejecución hasta que finaliza, conservando su historia en archivos magnéticos, con el objetivo principal de agilizar la actividad de los funcionarios involucrados.

Los trámites según su naturaleza son asignados a las dependencias de la Primera y Segunda Delegatura, Oficinas Asesoras, Secretaria General, Dirección Administrativa y Dirección de Informática y Estadística.

La base de datos de trámites y registro permitirá:

- La agilización y mayor eficiencia en la ejecución de los trámites.

- La centralización de la información para su oportuna y ágil consulta desde cualquier dependencia.

- El control del trámite desde que se genera hasta que finaliza. Se pueden generar por ejemplo controles de trámites por funcionario, por tipo de trámite, por entidad, etc.

- La generación automática de registros.

- La generación de un banco de datos útil como soporte a los otros subsistemas de información.

- La reducción del "papeleo" que implica la tramitología actual.

- El seguimiento personalizado a los diferentes solicitantes de trámites.

Para la correcta operación de esta base de datos, se localizarán terminales en puntos donde se les facilite a todas las dependencias la entrada y/o consulta de los datos. Deberá capacitarse al personal de la sección de radicación y a los responsables de cada dependencia en el manejo de los equipos.

Los microcomputadores serán utilizados como soporte a este subsistema en la elaboración de certificaciones, resoluciones, control de multas y sanciones principalmente.

BASE DE DATOS DE CONTROL DE LA CONCENTRACION

Se identificó como un subsistema de información nuevo para la entidad, orientado a velar por el cumplimiento de las normas en lo

correspondiente a la concentración de la propiedad y del crédito.

Esta base de datos se alimentará con información sobre sociedades, personas naturales, adquisiciones de acciones, operaciones de crédito y garantías.

Los tipos de resultados esperados incluyen informes de tipo general sobre accionistas directos, beneficiarios directos del crédito, informes para investigación de beneficiarios reales de la propiedad y del crédito, así como informes por entidad y beneficiario.

Dentro de los beneficios esperados podemos contar:

- Mejorar la calidad de la vigilancia e inspección a las entidades.
- Generar la base de datos de la llamada "central de riesgo".
- Consulta oportuna a la información sobre propiedad y crédito.

OTRAS BASES DE DATOS

Adicionalmente a las bases de datos anteriormente mencionadas, describimos brevemente aquellas que por razones de prioridad o por su naturaleza no son tan importantes para cubrir los objetivos de este artículo.

JURIDICA

El objetivo principal es la compilación de las normas y textos legales relacionados con el sector financiero, permitiendo su rápida consulta en las diferentes dependencias y la unificación de conceptos de los abogados.

ADMINISTRATIVA

Pretende agilizar y mejorar los procesos administrativos de la entidad creando un sistema integrado que permita el flujo de información entre las áreas de recursos humanos, financieros y físicos de la institución. Para el efecto se cuenta con paquetes de Nómina, presupuesto, personal, almacen, pagaduria, y control de resoluciones principalmente, los cuales podrán comunicarse a través de una red local de micros conectada al equipo central.

INFRAESTRUCTURA TECNOLOGICA

CRITERIOS DE SELECCION

Dentro de la política general del Ministerio de acogerse a los últimos adelantos tecnológicos y metodologías de desarrollo, se tomó como criterio principal el "software" y no el "hardware" como había sido tradicional en la entidad. Se añadió como componente de evaluación la portabilidad del "software" y la facilidad de contar con herramientas de última generación para desarrollo. Fundamentalmente, por estas razones, se decidió que el sistema operacional debería ser UNIX.

Con base en las particularidades de la entidad y los criterios mencionados con anterioridad, se definió que el manejo de los datos lo asumiría un manejador de bases de datos relacional que ofreciera generador de aplicaciones y reportes, buenas facilidades de protección y seguridad, e interfaz con lenguajes de alto nivel como "C" principalmente.

Se requería además del "software" y "hardware" para comunicarse con los computadores de las entidades financieras y con los microcomputadores de la entidad.

Se determinó que la adquisición de microcomputadores debería seguir la línea de estandares del mercado en ese momento y por lo tanto se adquirirían microcomputadores IBM o compatibles funcionando bajo el sistema operacional D.O.S. con paquetes de hojas electrónicas, bases de datos y procesadores de palabra entre otros.

Con estos criterios de selección, y los relacionados con las cantidades requeridas, durante el año 86 se adquirieron los siguientes elementos:

SOFTWARE

Se adquirió un sistema operacional UNIX compatible en un 100% con el Sistema V versión 2 de AT&T y en más de un 90% compatible con el de Berkeley 4.2.

ORACLE fuè el manejador de bases de datos escogido. Ofrece facilidades de cuarta generación para desarrollo de aplicaciones, generador de reportes, interfaz con "C" y componente para manejo de bases de datos distribuidas principalmente. ORACLE es un producto que actualmente se ofrece para multiplicidad de ambientes, marcas y familias de equipos (desde MICROS hasta MAINFRAMES).

Adicionalmente se comprò el "software" requerido para microcomputadores.

HARDWARE

Se adquirió un Superminicomputador modelo MV 8000 II producido por DATA GENERAL CORPORATION, con 8 Mbytes en memoria principal, aproximádamente 1800 Mbytes en discos fijos, dos unidades de cinta de 1600-800 B.P.I, dos impresoras ràpidas de 600 L.P.M. cada una, ocho lineas de comunicación asincrónicas/sincrónicas y cincuenta terminales monocromáticas.

Se han adquirido hasta el momento 20 microcomputadores IBM o compatibles, con memoria principal no inferior a 512 Kbytes, discos duros con capacidades de almacenamiento entre los 10 y los 30 Mbytes, unidades de diskette de 360 y 1200 Kbytes.

Adicionalmente se adquirieron 14 impresoras de caracteres con velocidades de impresión entre los 160 y los 300 cps, con capacidad para gràficas y opción de calidad, dos redes locales, y dos unidades de Back - up.

Se espera adquirir en un futuro próximo 20 microcomputadores màs con sus periféricos.

METODOLOGIA

Para la revisión de la especificación de los diseños se han utilizado los diagramas ENTIDAD - RELACION, diccionario de datos, elaboración de prototipos ràpidos, lo que ha permitido refinar la interfaz con el usuario, estructura de archivos y definición de programas.

Las estructuras de datos fueron llevadas a tercera forma normal, se eliminaron sinónimos y homónimos, identificando adicionalmente rutinas comunes y procesos de control y administración de las diferentes bases de datos.

ESTADO DEL PROYECTO

La implantación de la base de datos de Contable y Estadistica se adelanta en una primera versión sin el plan de cuentas generalizado.

utilizando los diferentes informes que actualmente maneja la entidad e implantando los procesos de lectura, validación, controles, informes y consultas.

La programación de la versión sin plan de cuentas de las bases de datos de Contable y Estadistica y la de Trámites y Registro será finalizada en el año de 1987.

La programación del subsistema de Concentración de la Propiedad y del Crédito se iniciará comenzando el año de 1988.

La sistematización del area administrativa se encuentrá bastante adelantada y es así como ya se tienen en operación los módulos de personal, presupuesto, y control de resoluciones.

En el area de la sistematización Jurídica se desarrolló un prototipo que se encuentra en su segunda versión. Se espera sea programada la base de datos en el año de 1988.

CONCLUSIONES

El esfuerzo en modernización que está haciendo la entidad no solo cubre el proyecto de sistematización sino que procura a través de la reestructuración administrativa darle un ordenamiento más funcional y adecuado para los tiempos de hoy, teniendo en cuenta que en las últimas décadas el sector vigilado ha multiplicado en varias veces su tamaño haciendo mas compleja y dificil la labor de vigilancia.

Todo esto reforzado con el hecho que las entidades finacieras conforman uno de los sectores más avanzados del país en el área de informática ya que iniciaron la labor de sistematización hace muchos años superando tecnológicamente a la entidad desde hace también mucho tiempo.

En lo correspondiente al proyecto de sistematización, la continuidad y el empeño ofrecido por cada uno de los que han intervenido en su desarrollo permite que hoy en dia se puedan apreciar los primeros resultados.

Es quizas contradictorio que la imposibilidad de haberse sistematizado con anterioridad, permitió que la entidad seleccionara tecnologías de bases de datos relacionales y herramientas de desarrollo poderosas que adicionàlmente ofrecerán la posibilidad de moverse a nuevos equipos o nuevos sistemas operacionales, todo esto incrementando la vida útil de las aplicaciones y quedando tecnológicamente más avanzada que muchas de las entidades vigiladas que comenzaron su

proceso de sistematización en las pasadas decadas.

Aprovechamos para agradecer a todas aquellas personas que directa o indirectamente han colaborado en las actividades relacionadas con el proyecto.

BASE DE DATOS DE LA DEUDA PUBLICA DE COLOMBIA (FENIX)

Por:
Eduardo Asmar G.
Alfredo Hernández C.
Alejandro Salamanca G.
Andrés Suárez R.

Ministerio de Hacienda y Crédito Público

Bogotà, Colombia, Julio de 1987

Presentado a la XIII Conferencia Latinoamericana de Informática.

El presente artículo hace una breve descripción de uno de los proyectos de sistemas que actualmente se adelantan en el Ministerio de Hacienda de Colombia, como es FENIX, la Base de Datos de Crédito Público.

FENIX comprende toda la información de deuda pública interna y externa, clasificada según su función en siete áreas: Registro, Proyecciones, Destinación, Presupuestal, Análisis Económico, Tramitación y Legal. A través de diversas aplicaciones que interactúan con la Base de Datos, se proveen herramientas de registro, análisis, control y proyección de la deuda pública nacional.

El proyecto FENIX está desarrollado bajo INGRES, un sistema manejador de bases de datos relacionales, y en un ambiente operacional UNIX; tanto el ambiente como las técnicas de desarrollo utilizadas colocan a FENIX a la vanguardia de los sistemas de información gubernamentales en Colombia y en Latinoamérica.

Por otra parte, el diseño de la Base de Datos de Crédito Público es de los más completos a nivel mundial, gracias a su conjunto de información, a la diversidad de aplicaciones y a las tecnologías de desarrollo, de "hardware" y "software" utilizadas.

BASE DE DATOS DE LA DEUDA PUBLICA DE COLOMBIA (FENIX *)

I. INTRODUCCION

La Dirección General de Crédito Público (D.G.C.P.) es una de las seis Direcciones Generales del Ministerio de Hacienda, encargada, según el Decreto 076 de 1976, entre otras funciones, de: tramitar y autorizar las solicitudes de crédito presentadas por el Gobierno Nacional y las entidades territoriales y descentralizadas, así como los préstamos que otorgue el Gobierno Nacional; controlar y vigilar la ejecución de los proyectos financiados con el crédito público; registrar y contabilizar el crédito del Gobierno Nacional y de las entidades territoriales y descentralizadas; atender el servicio de la deuda pública nacional.

Para el ejercicio de dichas funciones, la D.G.C.P. se ha enfrentado a volúmenes crecientes de información, y por consiguiente a la necesidad de resolver el problema de sistematización de dicha información. Con esto último como objetivo se inició en mayo de 1985 un proyecto consistente en el diseño y montaje de una Base de Datos de Crédito Público (hoy llamada FENIX), que realice las siguientes tareas:

- Registrar los contratos de crédito externo e interno del país, y de sus desembolsos y pagos;

- Llevar información del estado de la deuda de cada entidad, proyecto, sector económico y categoría de inversión frente a cada organismo nacional o internacional de crédito;

- Facilitar a los funcionarios la programación y ejecución presupuestal de los créditos y suministrar herramientas que ayuden al análisis del estado de la deuda pública nacional a todo nivel;

- Dotar de herramientas para la proyección de los desembolsos y del servicio de la deuda, así como del movimiento presupuestal;

- Proveer mecanismos de alerta para que el servicio de la deuda se haga oportunamente;

(*) A raíz del incendio del 24 de Abril de 1987 en las oficinas de la D.G.C.P., y que destruyó los equipos de computación de la División de Sistemas, la Base de Datos de Crédito Público (y el proyecto) cambió su nombre de B.D.C.P. al definitivo de FENIX, porque la información y los programas se recuperaron en su totalidad de los discos, que ya se habían dado por perdidos. Es decir, la Base de Datos renació, como el Ave Fénix, de sus propias cenizas.

- Ejercer control en la tramitación de nuevos contratos, de pagos y desembolsos;

- Mejorar la capacidad de negociación del país frente a fuentes externas de crédito.

Dicho proyecto se encuentra en una etapa avanzada de desarrollo (buena parte de sus módulos se encuentra en producción) y constituye el fin a más de una década de proyectos e intentos fallidos para dotar a la D.G.C.P. con una infraestructura acorde con el desarrollo de la informática. Para lograr esto, se ha contado con una política tecnológica y un ambiente diferente al que había sido tradicional en el Ministerio en los últimos lustros.

El nuevo ambiente incluye:

- Equipo computacional y humano **no dependiente** del Centro de Información y Sistemas (CIS) del Ministerio, que es la entidad de soporte de sistemas para todo el ministerio, pero que por razones de prioridades y equipos ha centralizado la gran mayoría de sus recursos en las aplicaciones de la Dirección General de Impuestos;

- Nuevas herramientas y tecnologías para desarrollo de "software";

- Uso de Sistemas Manejadores de Bases de Datos Relacionales (SMBR);

- Garantía de continuidad del proyecto frente a cambios de administración.

En las dos secciones que siguen se describe FENIX, la Base de Datos de Crédito Público. La primera sección presenta la estructura global de FENIX, sus archivos y procesos relacionados. La segunda sección describe brevemente el ambiente operacional bajo el que trabaja FENIX.

II. DESCRIPCION DE FENIX

El modelo teórico que se utilizó para el diseño fué el llamado "modelo entidad-relación" de Chen, el cual se implantó en un sistema relacional (INGRES). Vale la pena definir los dos conceptos fundamentales del modelo de Chen, es decir, las entidades y las relaciones. Una **entidad** representa un objeto distinguible dentro del mundo que se pretende modelar (en este caso la Dirección General de Crédito Público), y una **relación** la información que resulta de la asociación de dos o más entidades. Desde el punto de vista del esquema relacional, tanto las "entidades" como las "relaciones" son tablas, que son llamadas relaciones y que están compuestas por filas (tuplas) y columnas (atributos).

1. Generalidades

La base de datos contiene un total de 52 tablas o relaciones en el estado actual, cada una de ellas con un promedio de 3000 tuplas. Adicionalmente se ha definido un total de 29 índices sobre las tablas.

2. Descripción de la Base de Datos

FENIX está dividida en siete grupos o áreas funcionales, que unidas cubren toda la información de Crédito Público y que están formadas por un conjunto de tablas relacionadas entre sí desde un punto de vista lógico.

A continuación se describen brevemente las áreas funcionales en las que se encuentra dividida la Base de Datos de Crédito Público:

- **Area de Registro:** Es el núcleo de la base de datos y contiene la información básica de todos los créditos y de todas las personas o entidades que tienen relación a través de ellos. Más específicamente se tiene información de créditos con sus saldos, entidades (prestamistas y prestatarios), monedas, desembolsos y pagos, almacenada en 19 tablas.

- **Area de Proyecciones:** Tiene las cifras concernientes a los intereses, plazos, amortizaciones y comisiones de cada crédito, así como las proyecciones para el presupuesto de desembolsos y pagos. La componen 8 tablas.

- **Area de Destinación:** Como su nombre lo indica, en esta área se encuentra la información (acumulada y en detalle) del destino que se da a la utilización de los créditos, según la localización geográfica, los objetos, los proyectos, las categorías de inversión BID-BIRF y los sectores económicos. Consta de 10 tablas.

- **Area Presupuestal:** Contiene datos sobre la incorporación y ejecución presupuestal de los créditos, además de los movi-

mientos de dinero entre artículos (traslados, recursos, etc.). Está conformada por 3 tablas.

Area de Análisis Económico: Agrupa la información relativa a los indicadores económicos y a las estadísticas de contrataciones y utilizaciones por sector económico. En esta área hay 3 tablas.

Area de Tramitación: En esta área se almacenan y controlan los trámites que debe seguir un crédito hasta el momento de su contratación. Incluye los pasos a seguir y los funcionarios encargados de cada paso, en 3 tablas.

- Area Legal: Contiene la información jurídica que se involucra en el crédito, como lo concerniente a contratos, cláusulas, reglamentación, normas y conceptos legales. Se compone de 6 tablas.

En los diagramas de las paginas siguientes se puede apreciar un esquema de la Base de Datos y una vista detallada del Area de Registro.

Sobre estas áreas se pueden resolver las consultas asociadas a los procesos normales y también consultas de carácter general, tales como:

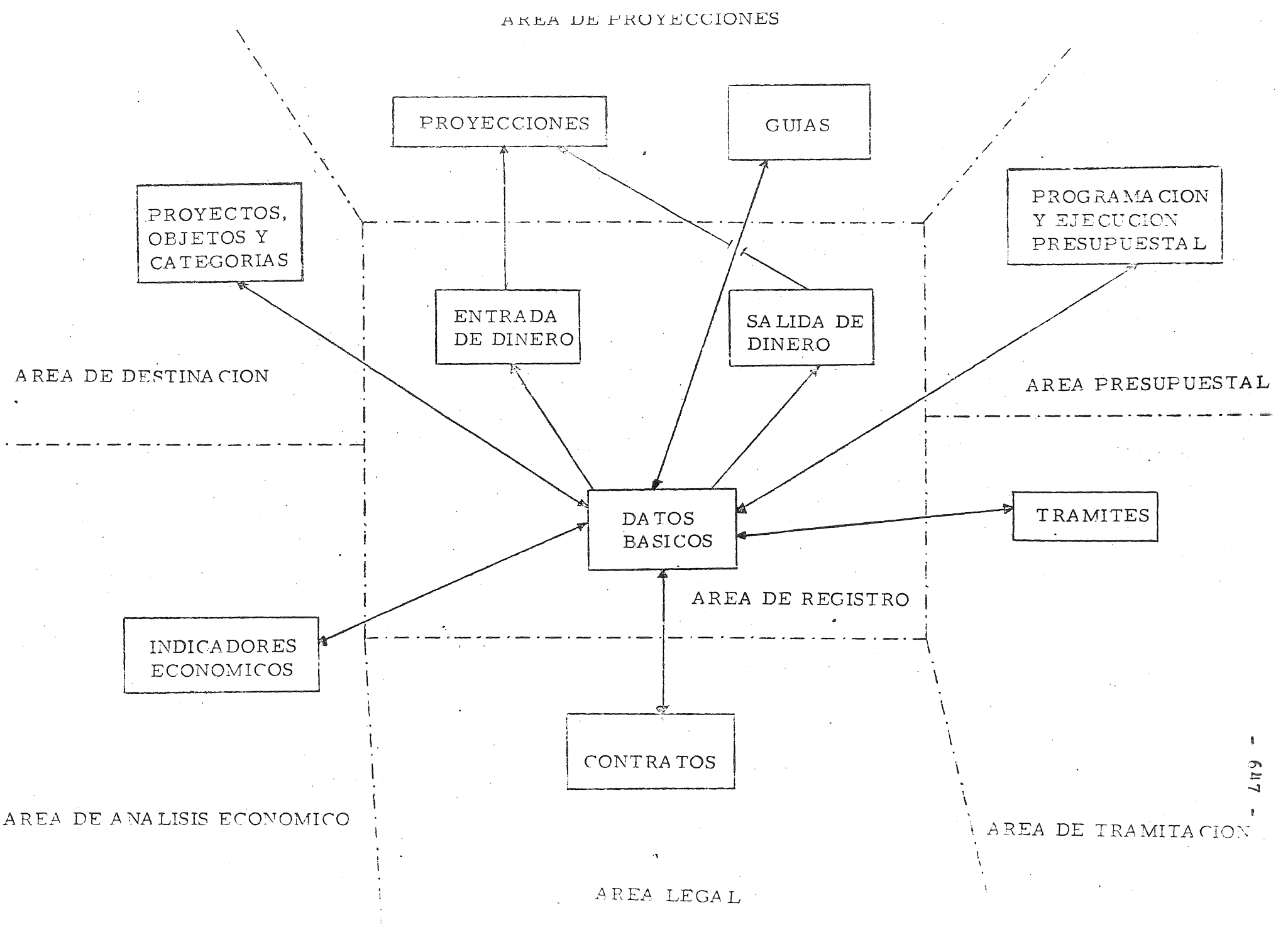
- Cuánto hay que pagar el mes entrante y en qué monedas?
- Se vence algún pagaré sobre algún desembolso que no se haya efectuado?
- Qué créditos se han contratado con Suecia y cuánto suman éstos en dolares?
- Cuánto se contrató el año anterior para educación?
- Qué entidades ejecutoras del orden Nacional han recibido préstamos para adquisición de equipos?
- Cuánto ha prestado el Japón para el sector eléctrico en los últimos cuatro años?
- A cuánto asciende (en Marcos Alemanes) los pagos que deben efectuarse en lo que resta del año al KFW por concepto de préstamos de defensa?

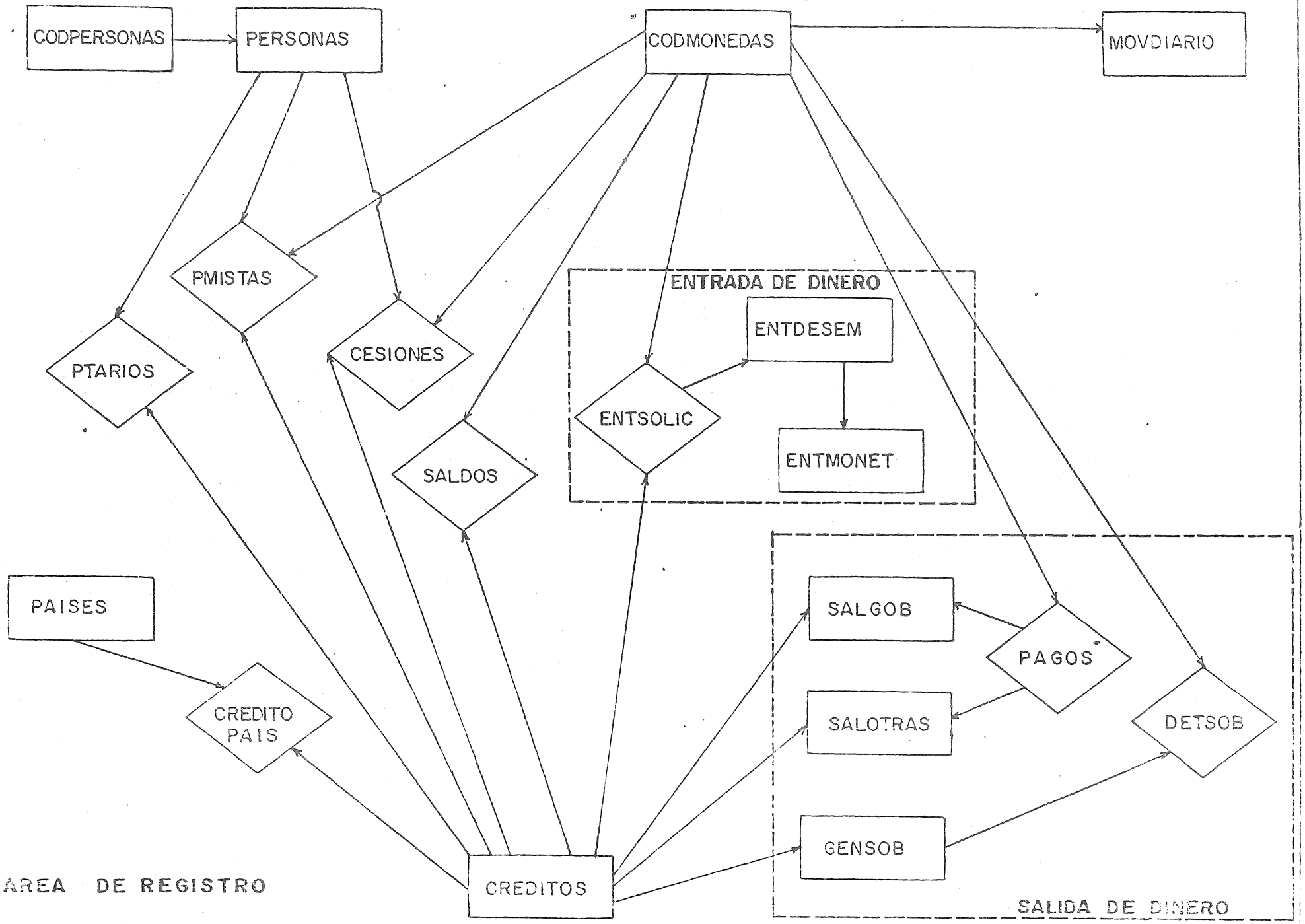
3. Descripción de procesos

Por un **proceso** se entiende un conjunto de programas y procedimientos, ya sean éstos manuales o escritos en un lenguaje de programación, que interactúan con un propósito común. Los procesos se dividen en cuatro grupos funcionales, de acuerdo a ese propósito:

- Procesos de carga inicial.
- Procesos normales.
- Procesos de administración.
- Procesos de seguridad.

A continuación se describe cada uno de éstos grupos.


AREA DE PROYECCIONES
PROYECCIONES
GUIAS
PROYECTOS, OBJETOS Y CATEGORIAS
PROGRAMACION Y EJECUCION PRESUPUESTAL
ENTRADA DE DINERO
SALIDA DE DINERO
AREA DE DESTINACION
AREA PRESUPUESTAL
DATOS BASICOS
TRAMITES
AREA DE REGISTRO
INDICADORES ECONOMICOS
CONTRATOS
AREA DE ANALISIS ECONOMICO
AREA DE TRAMITACION
AREA LEGAL


CODPERSONAS
PERSONAS
CODMONEDAS
MOVDIARIO
PMISTAS
PTARIOS
CESIONES
SALDOS
ENTRADA DE DINERO
ENTDESEM
ENTSOLIC
ENTMONET
PAISES
CREDITO PAIS
SALGOB
PAGOS
SALOTRAS
DETSOB
GENSOB
CREDITOS
AREA DE REGISTRO
SALIDA DE DINERO

3.1 Procesos de carga inicial.

En este grupo se encuentran aquellos procesos necesarios para manejar la carga masiva de los crèditos contratados hasta el momento. El objetivo es poblar FENIX con datos confiables de una manera rápida y efectiva, procurando filtrar el mayor número de errores, antes de que se inicie la producciòn normal.

3.2 Procesos normales.

A este grupo pertenecen los procesos que reflejan las diferentes funciones de Crèdito Pùblico y constituyen la columna vertebral del sistema. Los procesos normales se agrupan a su vez de la siguiente manera (ver gràfico de la pàgina siguiente):

1. Registro y Control.
2. Presupuesto.
3. Planeaciòn y proyecciones.
4. Legal.

3.2.1. Registro y Control.

Comprende todos los procesos que permiten hacer el seguimiento de la vida de un crèdito a travès de sus diferentes etapas:

- Solicitud.
- Contrataciòn.
- Ejecuciòn.
- Finalizaciòn.

Los mòdulos que lo componen son los siguientes:

a. Tramitaciòn.

Mediante este mòdulo se registran y controlan los trámites de los crèditos desde el momento de la solicitud hasta el momento de la contrataciòn, permitiendo la consulta y actualizaciòn de los mismos.

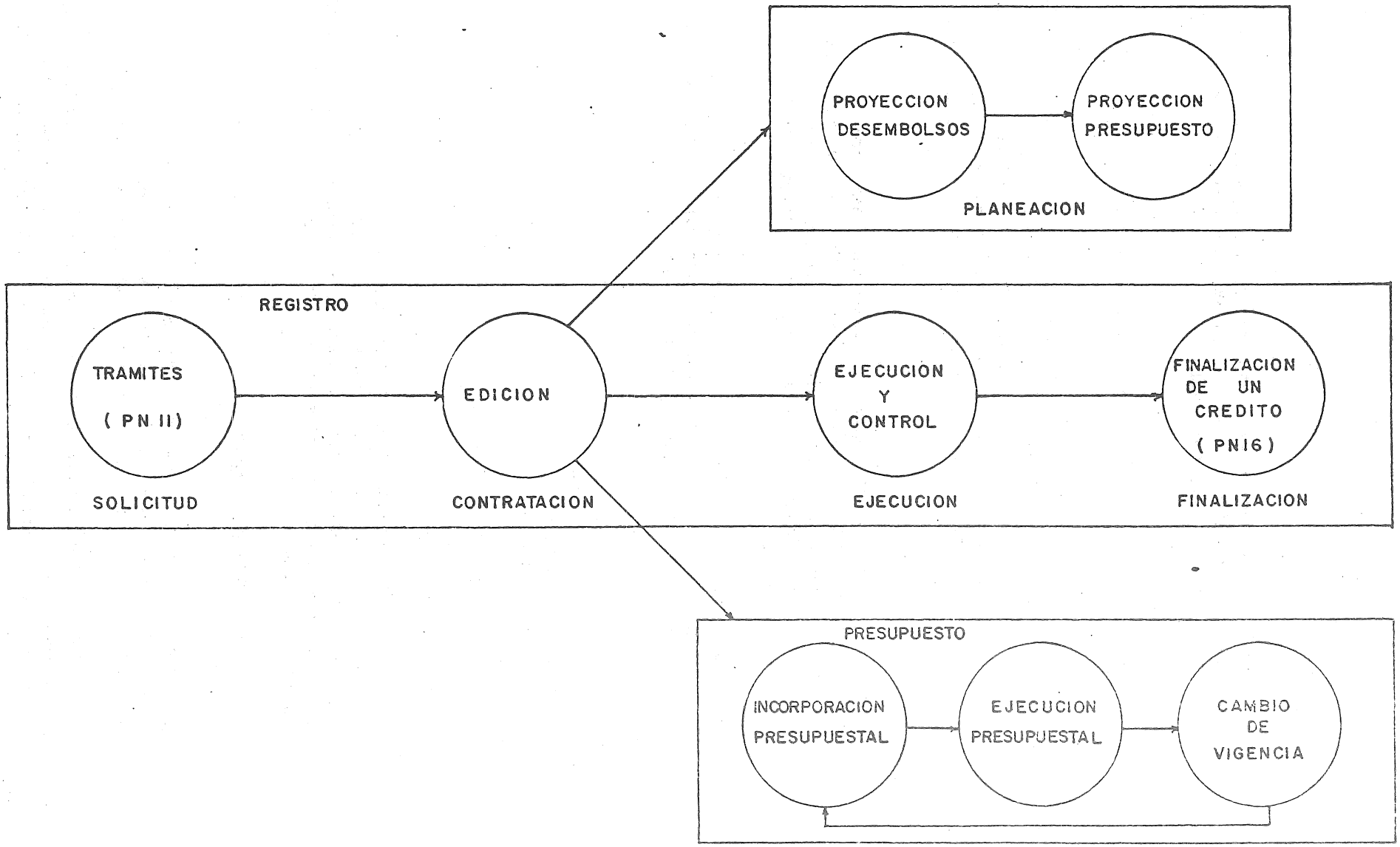
Aquí se registran las etapas o pasos de cada crèdito y el funcionario correspondiente que diligencia cada una. Además se verifica que un crèdito que se vaya a contratar haya cumplido con cada una de èstas.

Este proceso permite tambièn la consulta de los trámites. Algunas de las consultas típicas que se resuelven son:

- Cuál es el estado actual de determinado crédito? En qué etapa de tramitaciòn se encuentra y cuáles son los pasos que faltan para su contrataciòn?
- Cuáles crèditos han sido solicitados?
- Qué funcionarios han diligenciado determinado crèdito?

PROCESOS NORMALES

(PNXX)



b. Edición.

En este módulo se encuentran los procesos que permiten la modificación y actualización de la información concerniente a un crédito o a ciertas tablas de FENIX. A su vez se subdivide en los siguientes procesos:

- Edición de créditos: Permite la adición, modificación y eliminación de créditos.

  Una vez que un crédito ha cumplido con todas las etapas de tramitación hasta el momento de su contratación, éste se adiciona a FENIX por medio de la opción de adición, en la que se pide información del crédito tal como prestamistas, prestatarios, destinación, montos y monedas contratados, condiciones financieras, etc.

- Movimiento diario de monedas: Este proceso permite la creación y modificación de monedas y tasas de interés, el registro diario de variación de cada una de las divisas y tasas de interés con respecto al dólar de los Estados Unidos y la consulta de las mismas.

- Edición de tablas: Permite la corrección y consulta de los datos de algunas relaciones que no son manejadas por los anteriores procesos de edición. Estas relaciones son:

  - Personas.
  - Países.
  - Localizaciones geográficas.
  - Sectores Económicos.
  - Objetos del crédito.
  - Funcionarios.
  - Etapas o pasos de tramitación.

c. Ejecución y control.

A este módulo pertenecen los procesos que sirven para seguir la vida del crédito a partir del momento de su contratación hasta la finalización del mismo, permitiendo el registro de desembolsos, pagos y cesiones de deuda que ocurran durante este período. Se subdivide en los siguientes procesos:

- Registro de desembolsos: Este proceso permite realizar dos funciones:

  1. Registrar la historia del desembolso desde el momento de su solicitud hasta la realización del mismo.
  2. Registro del desembolso mismo una vez que se ha confirmado.

  Además de los datos básicos de un desembolso, tales como crédito, fecha, moneda y monto, se pide información adicional acerca de la destinación del dinero, por entidades ejecutoras,

proyectos, sectores económicos, etc., con el fin de actualizar directamente las tablas del Area de Destinación. Algunas consultas típicas son:

- Desembolsos solicitados.
- Desembolsos de un crédito en un período dado.
- Destinación de un desembolso por entidad, sector económico o proyecto.
- Porcentaje monetizado de cada desembolso.

- Registro de pagos: Este proceso permite el registro de los pagos y sobrantes generados para cubrir cada uno de los créditos.

  En este proceso como en el anterior se permite el registro del pago como la historia del mismo. Una vez que un pago se efectúa, se afectan los saldos de los créditos directamente. Algunas consultas típicas son:

  - Pagos efectuados en determinado período, discriminados por concepto de amortización, intereses y comisiones.
  - Estado de un pago determinado.
  - Pagos efectuados para determinado crédito.

- Participación y cesiones de deuda: El objetivo de este proceso es actualizar y mantener los porcentajes y montos de participación de los prestamistas de un crédito.

  Mediante este módulo se permite básicamente el registro de una cesión de deuda, operación que consiste en que uno o varios prestamistas (cedentes) ceden parcial o totalmente parte de su participación del crédito a otros prestamistas (cesionarios); el proceso se encarga de adicionar los nuevos cesionarios como prestamistas del crédito y modificar los porcentajes de participación de los prestamistas involucrados en la operación. Debido al desfase que existe entre la fecha de una cesión y el registro de la misma, los pagos y desembolsos efectuados durante este período son corregidos bajo los nuevos porcentajes de participación.

d. Finalización de un crédito.

En este proceso se maneja el caso en que la vida de un crédito culmina normalmente (se han efectuado todos los pagos y desembolsos del mismo).

Este proceso se encarga de eliminar el crédito a través de las diferentes relaciones en que se encuentre, excepto en la tabla de créditos, donde se activa el indicador de culminación, para poder tener un registro de la existencia del mismo.

3.2.2. Presupuesto.

En este grupo se encuentran todos los procesos necesarios para controlar la utilización del monto presupuestado para cada crédito por año.

A continuación se describe cada uno de los módulos que componen este grupo.

a. Incorporación y ejecución presupuestal.

Este proceso permite controlar el gasto del monto presupuestado de cada artículo. Un artículo es un código que identifica presupuestalmente a un crédito, asignado por Crédito Público.

El control del gasto del monto presupuestado para cada artículo se hace registrando los movimientos que ocurren durante el año. Un **movimiento** es alguna de las siguientes operaciones:

- Incorporación inicial.
- Acuerdos.
- Anticipos.
- Traslados crédito.
- Traslados contracrédito.
- Adición presupuestal.
- Resolución sobrantes.
- Adición de traslados presupuestales.
- Ordenes de cancelación.
- Decretos contracrédito.

En cada uno de estos movimientos se registran y actualizan los saldos de cada artículo directamente. Algunas consultas típicas son:

- Movimiento total de determinado artículo.
- Saldos totales por artículo.
- Total apropiado o incorporado por artículo.

b. Cambio de vigencia presupuestal.

Este proceso realiza la renumeración de los artículos de presupuesto una vez que ha cambiado la vigencia presupuestal; permitiendo la adición, modificación y eliminación de cada uno de ellos. Además, elimina todos los movimientos del año transcurrido, dejando listas las relaciones de presupuesto para la iniciación de un nuevo año presupuestal.

3.2.3. Planeación y proyecciones.

En este grupo se encuentran aquellos procesos que sirven para las operaciones de planeación y proyección de la deuda.

a. Proyección de desembolsos.

Permite generar los desembolsos que se proyectan para cada crédito, que luego serán utilizados por el proceso de Proyección de Presupuesto.

b. Proyección de presupuesto.

Permite calcular los montos y fechas de pago que deben realizarse por concepto de amortizaciones, intereses y comisiones en un período dado, con tasas de cambio, interés y porcentajes de devaluación escogidos por el usuario.

Este proceso calcula los pagos de servicio de la deuda a partir de los desembolsos y saldos de los créditos (generados por el proceso anterior). Estos pagos se calculan en moneda original, en dólares y pesos. Algunas consultas típicas son:

- Total de pagos por crédito, discriminados por concepto de amortización, intereses y comisiones para determinado año.
- Totales por trimestre.

3.2.4. Legal.

Los procesos de este grupo están en etapa de especificación.

3.3. Procesos de administración y procesos de seguridad.

En estos grupos se encuentran los procesos orientados a la organización y mantenimiento de la integridad de la Base de Datos, a garantizar su seguridad, a mantener los archivos con sus estructuras óptimas. También están los procesos de "backup" y recuperación.

III. AMBIENTE DE DESARROLLO

La Base de Datos FENIX opera bajo la siguiente configuración de "software" y "hardware":

1. "Software":

- Sistemas operacionales CENTIX (UNIX sistema V versión 2), y BTOS.
- INGRES, sistema manejador de bases de datos relacional, versión 3.

La decisión de implantar el proyecto de la Dirección General de Crédito Público en un Sistema Manejador de Bases de Datos Relacional (SMBDR) se tomó fundamentalmente por la flexibilidad que ofrece un sistema de este tipo y por su adecuación al propio proyecto FENIX.

Entre las ventajas de un SMBDR se pueden destacar las siguientes:

- El tratamiento uniforme que se da a la información: cualquier objeto se visualiza como una **tabla.**

- La llamada "independencia de los datos" que se refiere al hecho de que existe una real separación entre el diseño lógico y el físico de una base de datos. Esto reduce el costo del desarrollo y mantenimiento de las aplicaciones. Por ejemplo en INGRES es posible modificar la estructura de las tablas y definir o eliminar indices sin necesidad de modificar los programas de aplicación, y obteniendo grandes mejoras de desempeño.

- Un SMBDR está más cerca del usuario final que cualquier otra aproximación (SMBD jerárquico o de red). Permite hacer consultas directas mediante el uso de un poderoso lenguaje de alto nivel (QUEL en el caso de INGRES) y utilizar herramientas cómo QBF (Query by Forms), mientras que en un modelo de red, por ejemplo, el usuario siempre tiene que usar programas de aplicación, los cuales hacen de interfaz (y a veces de barrera) entre él y sus datos.

- Las dos ventajas anteriores llevan a la facilidad que posee un SMBDR de responder "queries ad-hoc", la cual es una de las necesidades principales de un sistema de información para Crédito Público.

2. "Hardware":

- Computador UNISYS XE-550-5, con arquitectura de múltiples procesadores especializados, los cuales corren sus propias copias de los sistemas operacionales
- Dos procesadores de aplicaciones (AP) Motorola 68020 de 16,7 MHz. Cada uno corre una copia de CENTIX.

- Procesadores de comunicaciones, de archivos y de terminales, cada uno con una copia de BTOS.
- Memoria principal de 8 MB.
- Memoria en disco de 540 MB.
- Veinte terminales inteligentes.
- Impresoras, unidades de cinta y microcomputadores.

El XE-550 soporta los sistemas operacionales CENTIX y BTOS. BTOS funciona como sistema operacional de soporte a CENTIX y provee servicios generales de manejo de archivos y comunicaciones.

La arquitectura de múltiples procesadores del XE-550 está diseñada para operar de manera que para el usuario es transparente la distribución de tareas entre las diferentes CPUs. Existe para ello una herramienta llamada **ICC** ("Inter-CPU Communication"), la cual provee un mecanismo de comunicación entre las tareas que corren en los diferentes procesadores. Dicho mecanismo consiste en bloques de "pedidos" y "respuestas" que se intercambian entre los sistemas operacionales que corren en los procesadores.

Los "kernel" de CENTIX que corren en los procesadores AP utilizan ICC para coordinarse entre sí y con BTOS. ICC permite también que "kernel" diferentes parezcan un solo "kernel". Una característica importante de ICC es que genera una relación **cliente-servidor** entre procesos, en el sentido de que un proceso (el servidor) presta un servicio a otro proceso (el cliente). Cada interacción de ICC está compuesta de un pedido y una respuesta. Un **pedido** es un mensaje de un cliente; una **respuesta** es el mensaje de retorno del servidor. Los procesadores se enteran de la llegada de pedidos y respuestas mediante interrupciones. Por medio del mecanismo ICC, es posible para un procesador AP delegar a otros procesadores el manejo de entrada/salida hacia terminales y hacia disco.

IV. COMENTARIOS

El ambiente arriba mencionado, i.e., tanto la arquitectura de "hardware" como la infraestructura de "software" de desarrollo es novedosa en América Latina, y el proyecto FENIX bajo este ambiente se encuentra a la vanguardia en cuanto a sistemas de información gubernamental se refiere. La implementación del diseño es mucho más amplia y ambiciosa que la mayoría de los sistemas de deuda a nivel internacional, los cuales reúnen alguna o varias de las siguientes características:

- Utilizan conjuntos menos detallados de información, reduciendo las posibilidades de análisis.
- Están diseñados por organismos prestamistas (como el BIRF o el BID), con el objetivo de hacer más estricto el control de su deuda en particular, y por lo tanto pierden generalidad frente a los demás aspectos de la deuda de un país.
- No utilizan tecnología de Bases de Datos; son programas de aplicación escritos hace más de diez años, en muchos casos elaborados en FORTRAN o COBOL.

La principal característica del diseño conceptual de FENIX es la **generalidad**. Esto le permite (y ha permitido) asimilar con relativa facilidad situaciones extraordinarias dentro del contexto de Crédito Público en Colombia, tales como una eventual refinanciación de la deuda externa.

Para el diseño y desarrollo se ha obtenido un equilibrio entre lo académico (metodologías de desarrollo de "software" y utilización rigurosa de las herramientas que provee UNIX) y lo práctico. En una entidad como Crédito Público no es posible ceñirse exclusivamente a patrones académicos debido a la permanente redefinición de prioridades y necesidades y por lo tanto, a la continua reespecificación de procesos y archivos (es decir, a la evolución constante del diseño de la Base de Datos). Sin embargo, se hace siempre uso de metodologías y modelos teóricos - tales como el modelo entidad-relación, el de prototipos rápidos para especificación interactiva (para los que INGRES ha resultado ser una herramienta excelente) -, y de las valiosas herramientas de UNIX para documentación y desarrollo.

Todos estos factores (ambiente y métodos) han constituído el elemento predominante en la gran aceptación y en la continuidad que ha tenido el proyecto FENIX.

AGRADECIMIENTOS

La realización de este proyecto ha sido posible gracias a la colaboración y el apoyo de muchas personas que directa o indirectamente han participado en alguna de sus etapas de desarrollo, y en particular, a la labor de los funcionarios de la Dirección General de Crédito Público.

BIBLIOGRAFIA

1. Banco Mundial. "Debt Systems Conference, Working Papers". External Debt Division, Economic Analysis and Projections Department, París, Mayo 29-31 de 1985.

2. Chen, Peter Pin-Shan. "The Entity-Relationship Model - Toward a Unified View of Data". ACM Transactions on Data Base Systems, vol. 1, No. 1, Marzo de 1976, pp. 9 - 36.

3. Ministerio de Hacienda y Crédito Público. "Diseño Conceptual del Sistema Integrado de Información de Crédito Público en Colombia", versión 1.0. Documento interno, Agosto de 1985.

4. Hernández, Alfredo, et al. "Proyecto de sistematización de la deuda pública de Colombia". Revista SISTEMAS de la Asociación Colombiana de Ingenieros de Sistemas, ACIS, No. 28 Julio - Septiembre de 1986.

5. División de Sistemas, Dirección General de Crédito Público, Ministerio de Hacienda y Crédito Público. "Base de Datos de Crédito Público, B.D.C.P.", versión 4.0 (FENIX). Documento interno, Junio de 1987.

APLICACION DE LOTUS 123 AL MANEJO DE DATOS DE RIEGO

Por RICARDO SANCHEZ

Dirección personal: Calle 19 Nº 31 - 33
Teléfono 359593
Cali - Colombia

Correspondencia: Apartado Aéreo 25636
Teléfono 392345
Cali - Colombia

El presente artículo se basa en la aplicación que se ha realizado del paquete LOTUS 123 en el manejo de los datos de campo, para el cálculo de riegos. La evaluación se basa en revisar el funcionamiento de ecuaciones, datos y funciones, en este caso para el cálculo de la evapotranspiración de los cultivos por uno de los métodos más utilizados mundialmente, el método de Penman. Se entiende la evapotranspiración, como la pérdida de agua en los cultivos por evaporación energética del agua del suelo y la transpiración físico-química de las plantas. El método de Penman combina el balance de energía con los aspectos meteorológicos. El proceso de evaluación forma parte del trabajo de investigación de pregrado que permitirá simular algunos parámetros necesarios para la programación de riegos y para lo cual la evapotranspiración es indispensable.

APLICACION DE LOTUS 123 AL MANEJO DE DATOS DE RIEGO

Por Ricardo Sánchez

Este es un trabajo sencillo que se basa en una aplicación práctica, que forma parte de la tesis de grado que elaboran el autor y los doctores Alfonso Díaz y Javier Carbonell en el CIAT.

La tesis de grado mencionada se basa en un " MODELO DE SIMULACION EN COMPUTADOR PARA TENSION DE HUMEDAD DEL SUELO (THS) CON FINES DE RIEGO ".

Para este trabajo se han utilizado varias subrutinas y funciones que se integran al programa definitivo. Tales subrutinas han sido desarrolladas en la Unidad de Estudios Agroecológicos del CIAT, para uso en cálculos meteorológicos, balance hídrico y estimación de la evapotranspiración por el método de Penman. La evaluación de las subrutinas y funciones es una labor ardua y demorada, por lo tanto es mejor evaluar su parte operativa (ecuaciones) con programas de Software Comercial. Esto representa un ahorro de tiempo en la elaboración de programas aplicados ó software especializado. La utilización de tales programas no solo tiene utilidad en investigación sino que pueden usarse directamente para el manejo de riegos.

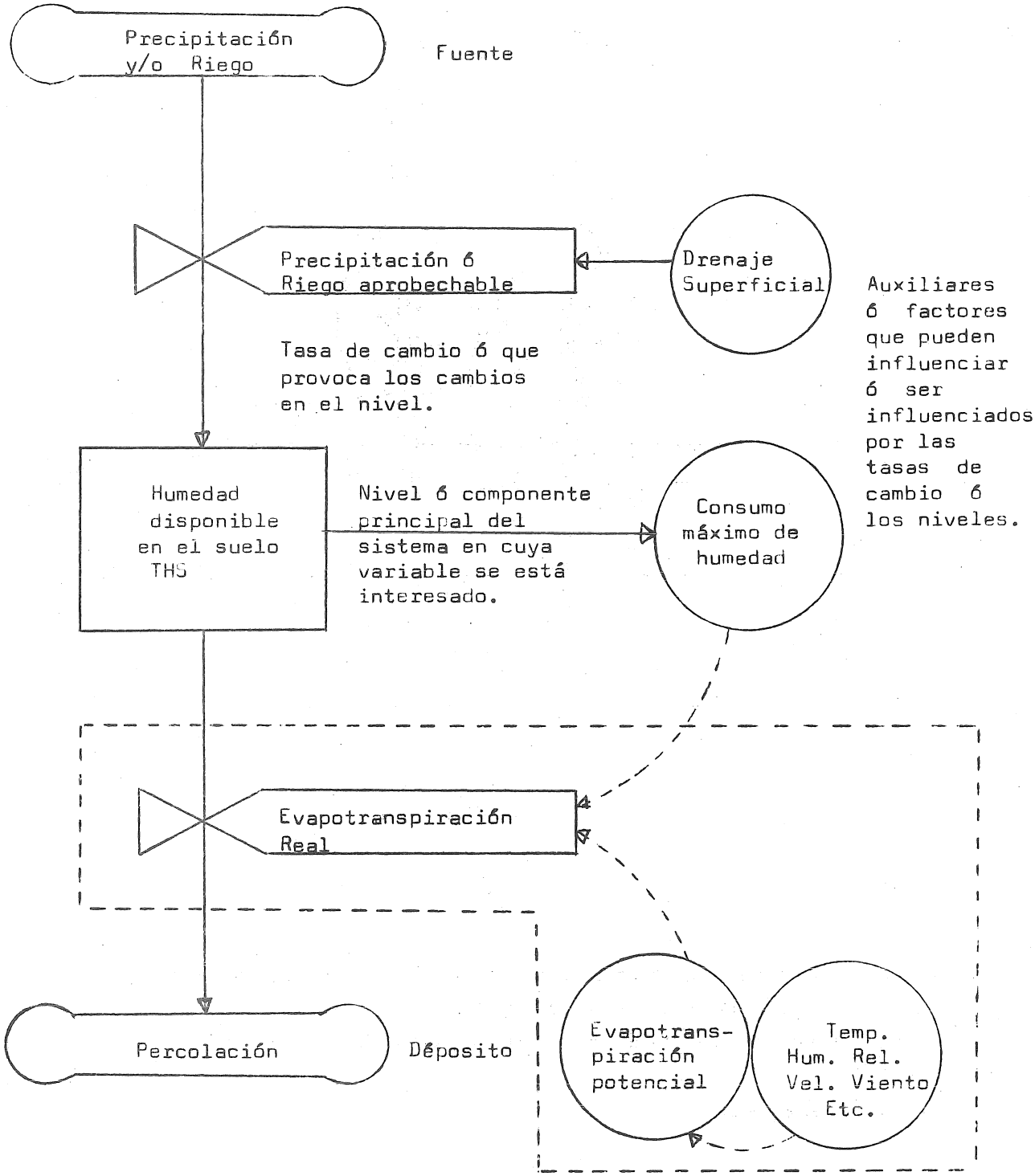
Según la metodología de Forrester (1960, 1968). El nivel en el modelo es el contenido de humedad disponible en el suelo lo que implica un valor de THS. A su vez dos tasas de cambio son la precipitación y/o riego y la evapotranspiración real. A su vez la evapotranspiración real depende de la potencial y del consumo máximo de humedad. Como se puede observar todo el modelo es un Balance Hídrico.

Es este caso se usa un Balance Hídrico sencillo desarrollado como un software especializado, llamado Watbal (Método de Keig y Mcalpine, 1974) (Evap., Prec., Cap. Suelo, Coef. Cult., Disponible, Drenaje ER/EP.) ER se igualará a EP solo si la humedad de suelo es igual ó superior a la capacidad de campo ó contenido de humedad óptimo, éste concepto es la base del método propuesto por la FAO para prácticas de riego.

Esta ponencia se concentra solo en una parte del diagrama ó balance, o sea en el cálculo de la evapotranspiración potencial usando la ecuación desarrollada por Penman que combina el balance de energía con los aspectos meteorológicos.

El Software Comercial utilizado permite calcular los valores buscados a partir de datos meteorológicos y ecuaciones. Entre los programas útiles para el cálculo y evaluación se encuentra el LOTUS 123 para el cálculo del requerimiento de agua por los cultivos, utilizando la ecuación de Penman para el cálculo de la evapotranspiración. En mi caso particular se me instruyó, para el cálculo de riegos, con el uso del valor de evapotranspiración, modificado por un factor de cultivo que varía según el cultivo y la región. Así debe aplicarse en el caso de usarse como único criterio para la programación del riego. Otros métodos utilizados son el Balance Hídrico, Tensiometros, Etc.

El modelo se puede describir en forma gráfica asi: Forrester(1960, 1968)



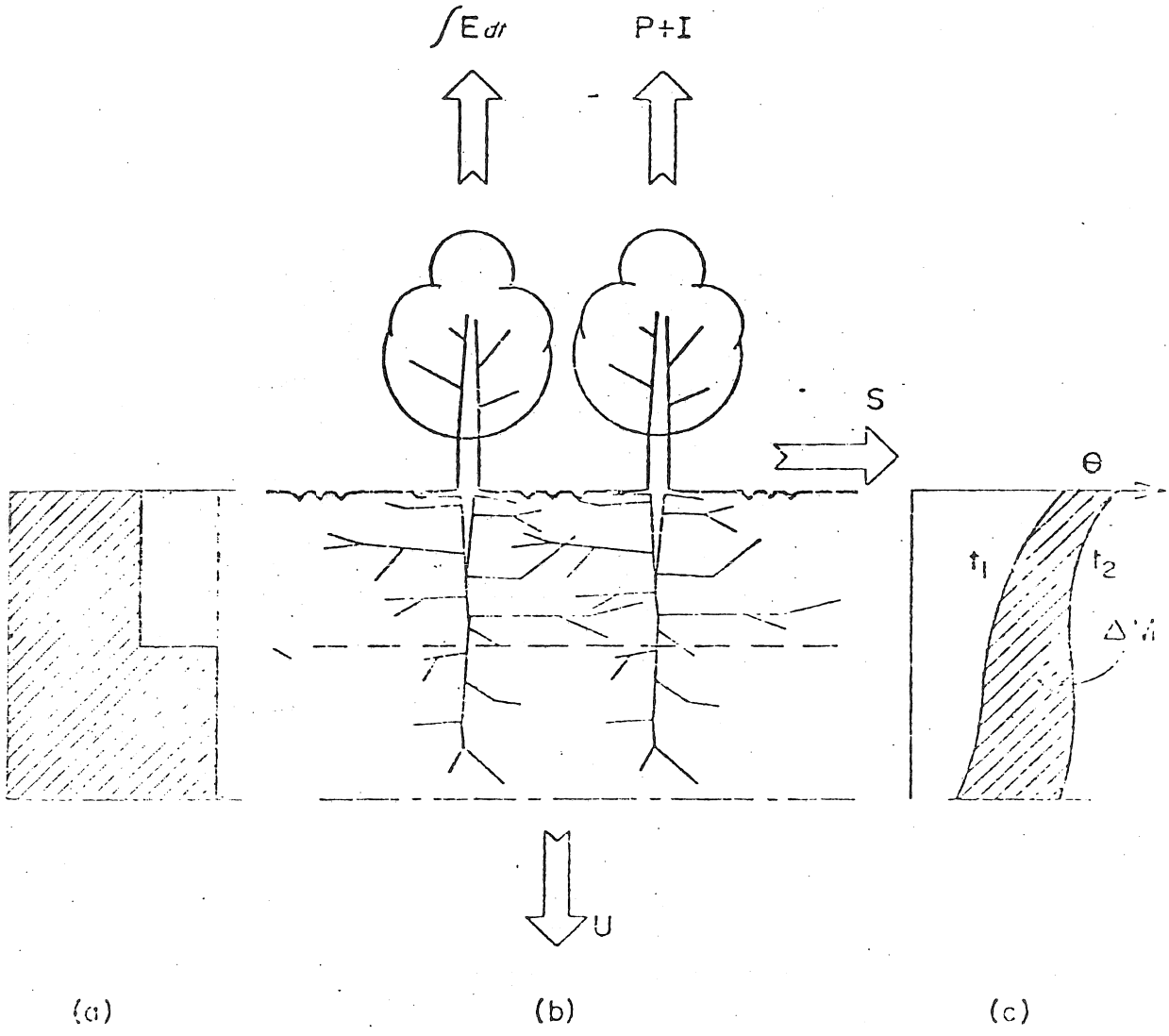


Figure 9.

(a) Amounts of water stored in root profile in a two layer soil profile.

(b) Water flows into and out of the root zone.

(c) Indication of how the storage of increased water in the root zone (ΔM) can be calculated.

Forrester (1968, 1969)

SIMULACION

COMPONENTES ESENCIALES:

Variables Exógenas ó de Entrada:	Variables no controlables, por ejemplo: Datos de evapotranspiración.
Variables Endógenas ó de Salida:	Por ejemplo: Respuestas del modelo ó estimaciones de la THS a partir de las variables exógenas.
Variables de Estado:	Variables intermedias entre las exógenas y endógenas. Por ejemplo: Condiciones del suelo como temperatura, contenido de humedad, histeresis.
Variables de Relación:	Coeficientes de ecuaciones ó leyes que describen la dependencia entre las variables endógenas y las de estado.

USO DE SOFTWARE COMERCIAL Y SU APLICACION AL MANEJO DE RIEGOS

En la presente ponencia se presenta el uso de la ecuación de combinación desarrollada por Penman (Allen, 1986), con algunas modificaciones posteriores. La ecuación combina el balance de energía con los aspectos meteorológicos. El software comercial disponibla, permite calcular los valores buscados, a partir de datos y ecuaciones, sin la necesidad de elaborar programas extensos y complicados.

Existen varios programas con diversas aplicaciones, que han sido usados por el autor para análisis de rendimientos y costos de proyectos agroindustriales, diseño hidraúlico de canales, etc.

La Ecuación de Penman

La ecuación de Penman tiene la siguiente forma general:

$$E = \frac{\Delta}{\Delta + \gamma} H_o + \frac{\gamma}{\Delta + \gamma} E_x$$

Donde,

E: Evapotranspiración (mm dia^{-1}), pérdida de agua en forma de vapor, por el suelo y la planta.

Δ : Pendiente de la curva de, presión de vapor saturado contra temperatura (mb $^oK^{-1}$), calculada en la temperatura media del aire. DELTA.

γ : Constante psicrómetrica (mb $^oK^{-1}$). GAMMA.

H_o: Radiación neta de onda larga y corta ($mmdia^{-1}$, equivalente agua).

E_x: Término aerodinámico de transporte de vapor ($mmdia^{-1}$).

$$H_o = (1-\alpha) R_s - \sigma T^4 (0.34 - 0.044 (e_2)^{\frac{1}{2}}) (0.10 - 0.90 \frac{n}{N})$$

Donde,

α: Albedo superficial.

R_s: Radiación solar incidente de onda corta ($mmdia^{-1}$, equivalente agua).

σ: Constante de Stephan - Boltzman ($2\ 10^{-9}$ $mmdia^{-1}$ $^oK^{-4}$)

T: Temperatura media del aire (^oK)

n/N: Relación de horas reales a posibles, de sol.

e_2: Presión de saturación de vapor (mb), a una altura determinada (Generalmente 2m) desde el suelo o la superficie de agua y a la temperatura del punto de rocio.

$$E_x = F(u) (e_{sat} - e_2)$$

Donde,

e_{sat}: Presión de saturación de vapor (mb), a una altura determinada desde el suelo o la superficie de agua y a la temperatura media del aire.

F(u): Función de la velocidad del viento.

$$F(u) = 0.263 (1+ 0.537u)$$

Donde,

u: Velocidad del viento (ms^{-1}) a 2m de elevación.

HOJA ELECTRONICA : LOTUS 123

El LOTUS 123 es uno de los software comerciales con la mayor aplicación en los últimos tiempos. Sin embargo su versatilidad permite otras aplicaciones diferentes a las comerciales, en este caso de Ingeniería Agricola.

Los pasos básicos para la carga y uso del programa se encuentran en el manual LOTUS 123. User's Manual (Kapor, M. y Otros, 1983).

Su funcionamiento básico es el de una hoja electrónica, base de datos y graficador.

Las caracteristicas del software, que se deben tener en cuenta para su aplicación son:

1. Entrada de compartimentos (cells), copiado, fórmulas, funciones matemáticas y estadisticas.

2. Graficado de datos y resultados.

3. Funciones de base de datos: arreglos (SORT), extracción (QUERY), distribuciones y regresión.

En el primer numeral, se puede anotar la versatilidad del LOTUS 123, al permitir una identificación lo suficientemente adecuada de los compartimentos, la copia de fórmulas y otros campos en la medida que se necesiten en cualquier dirección de la hoja electrónica (ver listado de compartimentos impreso por el LOTUS 123); Por ejemplo, en el caso del campo J7, se efectúa un cálculo previo, cuyo resultado final aparece ubicado en ese campo, como dato, al igual que los datos de K7, L7, M7, N7, O7. En estos casos el LOTUS 123 procesa esta información como VALUE.

En el caso de los textos de encabezamiento de las columnas, LOTUS 123 procesa esta información como LABEL, permitiendo una serie de operaciones de organización de los textos, diferente e independiente del manejo de datos.

En el caso de las fórmulas utilizadas su sintaxis es:

$$H_o = (1 - \alpha) R_s - \sigma T^4 (0.34 - 0.044 (e_2)^{\frac{1}{2}}) (0.10 + 0.90 \frac{n}{N})$$

En LOTUS 123 se subdivide en:

$$\sigma T^4 (0.34 - 0.044 (e_2)^{\frac{1}{2}})$$

P7: 5,67 * 0,00000001 * (J7 + 273,15)^4 *((0,34 - 0,044 *K7^(½)))*0,00001*24*60*60

$$(0.10 + 0.90 \frac{n}{N})$$

R7: 0,1 + 0,9 * Q7 / Y 3

El producto de estas dos subdivisiones :

S7: (1- $ Z $ 2) * M7 - P7 * R7

Aparece bajo el encabezado:

S5: ^ Rad. Neta.

Otra ecuación utilizada es:

$$F (u) = 0.263 (1 + 0.537 u)$$

Aparece, con una conversión de unidades necesaria:

U7: 0,263 *((1+ 0,537* N7 * 1000 / 24* 60* 60)))

A partir de estos cálculos se puede obtener:

$E_x = F(u)\ (\ e_{sat} - e_2\ )$

Como,

7: + U7 * (07/100) /K7* (1-07/100)* 24* 60* 60* 0,0001

donde se realiza una transformación implicita en la ecuación para utilizar otros datos, alimentados al programa.

Las funciones;

T7: 2,5003 - 0,002379 * J7

V7: 54055/((((0,9308+(10000/Q2* (1+10000/ Q2)))* (1051 - J7))))

W7: 43,74 + (3,12 + (0,0513 + 0,00223 * J7) * J7) * J7

fueron desarrollados por el doctor **Peter Jones PhD** de la **Unidad de Estudios Agroecológicos del CIAT** y están en proceso de evaluación en este trabajo de investigación.

Finalmente las ecuaciones:

$$\frac{\Delta}{\Delta+\gamma}\ H_o\ ,$$

Se representa en

Z7 : + S7 / X7 ;

$$\frac{\gamma}{\Delta+\gamma}\ E_x$$

En

AA7: + Y7 / X7 * V7 / W7

Y, finalmente,

$$E = \frac{\Delta}{\Delta+\gamma}\ H_o + \frac{\gamma}{\Delta+\gamma}\ E_x$$

En

A B 7: + Z7 + AA7

Se presenta una impresión de los cuadros de resultados en el estilo sin formato (UN_FORMATTED) de la instrucción PRINT.

CONCLUSIONES Y RECOMENDACIONES

El uso de software comercial en vez del especializado para agricultura, tiene como ventajas los múltiples usos que puede tener el programa, en este caso LOTUS 123.

La elaboración de programas en lenguajes de computación (BASIC, FORTRAN, ETC.) se puede evitar utilizando el software apropiado, poniendo al alcance de todos las técnicas de computación.

En el cálculo de los requerimientos de agua por los cultivos, LOTUS 123 facilita el cálculo y recálculo de las ecuaciones necesarias, un manejo adecuado de los datos y rapidez en la obtención de probables resultados; permitiendo descartar o utilizar ecuaciones y datos, después de una evaluación rápida.

REFERENCIAS

Allen, R.G. 1986. A Penman for all Seasons. Journal of Irrigation and Drainage Engineering. ASCE. 112 (4) : 348 - 368.

Forrester, J.W. 1960. Principles of Systems. Cambridge, Wright - Allen.

Forrester, J.W. 1968. Industrial Dynamics. Cambridge, Mass. MIT Press.

Keig, G. y McALPINE, J.R. 1974. WATBAL: A Computer System for the Estimation and Analysis of Soil Moisture Regimes From Simple Climatic Data. Znd ed. CSIRO, Australia.

Kapor, M.; Sachs, J. y Otros. 1983. LOTUS 123 User's Manual. Lotus Development Corporation y The CSA Press U.S.A.

SISTEMA UNIVERSITARIO DE INFORMATICA JURIDICA

Jesús Santos Amaya
Instituto Colombiano para el Fomento de la Educación Superior
ICFES
Apartado Aéreo 6319
Bogotá - Colombia

El Instituto Colombiano para el Fomento de la Educación Superior, ICFES, consciente de la importancia de la informática como instrumento de desarrollo nacional, ha conformado el Sistema Universitario de Informática Jurídica- SUIJ, considerada hasta ahora como el desarrollo mas importante, en lo que a Informática Jurídica se refiere, en LatinoAmérica. Este trabajo ha sido realizado por el ICFES, a través de la Secretaria General y de la División de Sistemas e Informática, contando con la colaboración estrecha de la Oficina de Planeación de la Universidad Externado de Colombia y el Centro de Investigaciones Socio Jurídicas de la Universidad de los Andes. El sistema permite procesamiento, control y recuperación de la información jurídica a nivel de Legislación, Jurisprudencia y Doctrina en las diferentes áreas de Derecho, de manera automatizada.

El Software del Sistema es entregado sin ningún costo a todas las instituciones públicas, instituciones de educación postsecundaria, bibliotecas e instituciones sin ánimo de lucro, que estén interesadas en montar su propio banco de datos y quieran contribuir al desarrollo integral del sistema.

Para tal efecto, las instituciones interesadas deben firmar un convenio con el ICFES, éste a su vez prestará la asesoría tanto técnica como jurídica que se requiera, a cambio de que la institución conforme un banco de datos en un área específica del derecho previamente acordada entre las partes.

El ICFES además de la asesoría técnico-jurídica, y de las demostraciones del banco de datos en sus instalaciones, ha realizado diferentes seminarios sobre el tema, en ciudades como Bucaramanga, Manizales, etc., el último de ellos en el mes de marzo en la Universidad Externado de Colombia, con el ánimo de establecer un trabajo de cooperación y coordinación que logre la consolidación del Sistema Universitario de Informática Jurídica que posteriormente formará el Sistema Colombiano de Informática Jurídica. Vale agregar, que países de CentroAmérica, además de Perú y Cuba, han manifestado su interés por montar el sistema desarrollado por el ICFES y han solicitado la asesoría correspondiente.

COMO ES EL SISTEMA UNIVERSITARIO DE INFORMATICA JURIDICA-SUIJ.

El SUIJ es un sistema automatizado que permite el procesamiento, control y recuperación de información en cualquier área del Derecho, a nivel de legislación, jurisprudencia y doctrina. El sistema ha sido desarrollado utilizando las últimas técnicas de

programación, es interactivo, modular, amistoso, permitiendo su consulta en línea, de manera conversacional, en forma sencilla, a través de menús, facilitando así, su consulta a personas sin ningún conocimiento de sistemas.

El sistema se complementa con el SCIB, Sistema Colombiano de Información Bibliográfica, que permiten tratamiento de toda información documental, convirtiendose en un gran sistema manejador de bases de datos. Sus características de desarrollo y transportabilidad, han permitido su conversión a diferentes marcas y modelos de equipos.

El sistema está conformado por cuatro (4) niveles, a saber:

- Documento Anfitrión o Bibliográfico: Puede ser una compilación de normas, una monografía o una publicación seriada.
- Documento Jurídico: Corresponde a cualquier tipo de norma ya sea: Constitución, Ley, Decreto, Resolución, Acuerdo, Ordenanza, Circular, Sentencia, Salvamento, Concepto, Artículo, Doctrina, etc.
- Unidad Informática: En legislación es el artículo en si mismo y en jurisprudencia y doctrina es el problema jurídico.
- Texto: En legislación se toma el texto completo de la norma, en los demás eventos, el extracto jurisprudencial o doctrina.

FUNCIONALIDAD DEL SISTEMA

El sistema se divide en dos grandes subsistemas, a saber:

- Sistema de Captura: Permite crear y modificar registros para cada una de las bases de datos que conforman el sistema. La captura de los datos para los niveles bibliográficos jurídicos y unidad de informática, se realiza en el formato SCIB, basado en el formato MARC (Maching Readable Cataloging), de la biblioteca del Congreso de los EEUU, estandarizado internacionalmente.

- Subsistema de Consulta: A través de el, el usuario podrá accesar el nivel de información que desee. Documento anfitrión, Documento jurídico, Unidad de Informática. Especificado el nivel, el usuario puede consultar por temas, autor, título, etc.

PROYECCIONES:

El ICFES se encuentra en el proceso de montaje de la Red Nacional de Comunicaciones para la Educación Superior, subred de la Red Nacional de Transmisión de Datos por Conmutación de Paquetes del Ministerio de Comunicaciones y su administrador será el Sistema de Información y Documentación para la Educación Superior-SIDES. A través de la mencionada red, se pondrá al servicio de la comunidad universitaria y de los investigadores en general, el Sistema Universitario de Informática Jurídica, el cual, contribuirá al desarrollo de la informática, en el área del derecho, conformando el Gran Sistema Nacional, urgente para el país.