

DisCo: Um Ambiente de Programação Distribuída com Configuração Dinâmica de Processos

JORGE DE ARAÚJO LIMA FILHO
PAULO ROBERTO FREIRE CUNHA

Universidade Federal de Pernambuco
Departamento de Informática
Caixa Postal 7851
50739 Recife-PE, Brasil
jalf@di.ufpe.br
prfc@di.ufpe.br

Resumo

Flexibilidade para modificar dinamicamente a configuração de sistemas distribuídos é uma característica fundamental para a implementação de sistemas robustos. Para atender devidamente às expectativas dos usuários, um ambiente de distribuído criado para construção de sistemas reais precisa oferecer recursos que garantam a flexibilidade, modularidade, robustez e dinâmica inerentes a sistemas reais. O ambiente DisCo oferece tais propriedades através das características do seu ambiente de programação e do seu ambiente de suporte à configuração dinâmica.

Abstract

Flexibility is an important property of distributed systems. Dynamic distributed systems configuration represents a crucial point in robust systems implementation. Distributed environments have to present resources which improve the flexibility, modularity, robustness and dynamism associated with real time systems construction. DisCo environment offers these properties throughout its programming and runtime supporting features.

Key Words

Distributed Systems, Flexibility, Modularity, Dynamic Configuration.

1 Introdução

Os avanços tecnológicos ocorridos nos últimos anos nas áreas de integração de componentes (VLSI), redes de computadores e teleprocessamento viabilizaram, não apenas sob o ponto de vista técnico, mas principalmente sob o ponto de vista econômico, a interligação cada vez maior de máquinas (computadores). Inicialmente, a interligação de máquinas trouxe consigo a possibilidade de compartilhamento de recursos. Atualmente, o horizonte de aplicação de computadores interligados expandiu de tal forma que é possível não apenas o compartilhamento de recursos, mas sobretudo o desenvolvimento de aplicações distribuídas. Desta forma, não tardou a surgirem os primeiros sistemas operacionais de redes e posteriormente os primeiros sistemas distribuídos[1]. A partir daí, intensificaram-se os esforços no desenvolvimento de linguagens para programação concorrente e distribuída, ao mesmo tempo em que começaram a ser estudados os primeiros problemas relacionados com “deadlocks”. Posteriormente surgiram preocupações com relação à estruturação de programas distribuídos (concorrentes) de forma a reduzir a complexidade associada com a programação de sistemas distribuídos. Isto levou à introdução de mecanismos para o fornecimento de modularidade (processos, módulos, recursos, etc) nas diversas linguagens de programação distribuída existentes. Até então, as linguagens de programação distribuída (CSP, SR, Ada, etc) permitiam a construção de sistemas distribuídos *estáticos*, ou seja, sistemas cuja estrutura permanece constante durante sua execução. Desta forma, propriedades importantes de sistemas distribuídos, como *flexibilidade e robustez*, não podem ser totalmente exploradas. Daí surgiram novos ambientes de programação distribuída que, explorando a flexibilidade e a robustez, permitem a construção de sistemas dinâmicos, ou seja, sistemas que podem ter sua estrutura modificada durante a sua execução. Assim como CONIC[2], Durra[3] e HPC[4], **DisCo** é um ambiente que permite e explora a configuração dinâmica na construção e evolução de sistemas distribuídos.

Os maiores problemas enfrentados pelos projetistas de ambientes distribuídos estão no fornecimento de mecanismos adequados para suportar a reconfiguração dinâmica de sistemas. Isto engloba desde comandos para especificar reconfigurações, até mecanismos de suporte que implementem reconfiguração dinâmica de forma segura e consistente [5]. Outro foco recente de pesquisas na área de sistemas distribuídos diz respeito ao desenvolvimento de novos modelos de gerenciamento para introdução dinâmica de mudanças em sistemas distribuídos [6, 7].

2 Ambiente DisCo

O ambiente **DisCo** foi concebido a partir de pesquisas realizadas pelo grupo de Redes de Computadores e Sistemas Distribuídos da Universidade Federal de Pernambuco (Brasil) [8, 9]. Este ambiente adota o modelo de programação a nível de configuração como metodologia para descrição, construção e evolução de sistemas.

DisCo é um ambiente para desenvolvimento de sistemas distribuídos com suporte à configuração dinâmica de processos. Este ambiente está sendo desenvolvido para permitir a modificação de sistemas de forma *dinâmica, segura e consistente*. O ambiente distribuído **DisCo** possui duas características principais que o diferenciam de outros ambientes existentes [2, 3, 4], a saber:

1. afeta o mínimo possível a estrutura do sistema durante o processo de reconfiguração, mantendo assim sua máxima funcionalidade (do sistema),
2. permite a especificação de modificações na estrutura do sistema ainda na construção do mesmo para o tratamento de possíveis falhas que venham a ocorrer.

2.1 Ambiente de Programação

Sistemas distribuídos, normalmente, são complexos, não apenas pelo seu tamanho, mas sobretudo pela variedade de interações existente entre os diversos módulos que os compõem. *Modularidade* é o princípio aplicado para reduzir a complexidade associada à construção de sistemas distribuídos, onde o sistema é sub-dividido em um conjunto de tarefas simples executando paralelamente e trocando informações para implementar a funcionalidade do sistema.

A propriedade essencial de sistemas distribuídos é a *flexibilidade*, porém muitas linguagens não permitem modelar esta propriedade de forma abrangente. Com este intuito, optamos por definir uma linguagem para programação de módulos e uma linguagem especial para definir a configuração de sistemas como um conjunto de módulos interconectados. A linguagem de configuração oferece recursos que permitem que módulos sejam inseridos e retirados dinamicamente durante a execução do sistema. Além de garantir a flexibilidade, esta solução torna a modularidade mais eficiente, pois estabelece claramente a relação entre as partes de um sistema (módulos).

O ambiente de programação de DisCo é formado por duas linguagens: a *Linguagem de Programação dos Módulos* e a *Linguagem de Configuração* que estabelece as conexões e permite a programação dinâmica. A utilização de linguagens distintas para a programação dos módulos e a construção do sistema facilita a descrição, compreensão e manipulação da estrutura do sistema. A construção e modificação de sistemas distribuídos é implementada traduzindo-se a sua descrição estrutural na criação e remoção de módulos e na criação, remoção e troca de conexões.

2.2 Linguagem de Programação dos Módulos

A Linguagem de Programação de Módulos [8, 10] é usada para descrever um módulo-tarefa que é uma unidade de programa sequencial¹ implementando a funcionalidade do módulo. Esta linguagem fornece *independência de contexto* permitindo a construção e compilação de módulos de software independentemente da configuração na qual eles executarão. Para isto, os comandos da Linguagem de Programação utilizados dentro de um módulo se referem unicamente à variáveis locais. Referências globais restringem a facilidade de inclusão e remoção de módulos, além de reduzir a flexibilidade de alocar módulos a processadores. A interconexão de módulos não faz parte da Linguagem de Programação, é exclusividade da Linguagem de Configuração. Desta forma, os módulos ganham *independência de interconexão*, característica fundamental para a *reusabilidade de software*.

A comunicação entre módulos é realizada por meio de uma interface local ao módulo. Comunicação através da chamada direta a outras entidades não é permitido, pois limitaria a flexibilidade de configuração lógica de sistemas. A interface de comunicação dos módulos é formada por um conjunto de portas, que constitui o único meio de comunicação de um módulo com outros módulos e com o ambiente externo. A porta é um nome local ao módulo, desta forma todas as referências feitas por um módulo são a nomes locais a eles. Assim, garantimos a *independência sintática* dos módulos e asseguramos a reusabilidade de software. As primitivas de comunicação oferecidas pela Linguagem de Programação possuem a mesma sintaxe e a mesma semântica tanto na comunicação local (dentro de uma mesma estação), quanto na comunicação remota (entre estações). Esta propriedade, *transparência de comunicação*, facilita a construção de um sistema permitindo que este seja desenvolvido e testado em uma única estação, e depois, distribuído nas demais estações da rede. Isto é possível, porque a transparência de comunicação garante que o comportamento lógico de um módulo seja exatamente o mesmo, independentemente da estação

¹O paralelismo ocorre a nível de módulos.

na qual irá executar. Desta maneira, o ambiente fornece mais flexibilidade no desenvolvimento de sistemas distribuídos.

2.3 Linguagem de Configuração

A Linguagem de Configuração [8, 11] é utilizada para especificar a estrutura de um sistema distribuído como um conjunto de módulos interconectados. O paralelismo acontece a nível de módulos, onde os módulos representam a unidade principal de estruturação de sistemas. Daí a necessidade de utilizarmos uma linguagem específica para manipular módulos, com mecanismos para programação dinâmica que permitam a reestruturação de sistemas para atender às mudanças do ambiente.

A especificação de um sistema distribuído consiste em quatro etapas distintas: *definição de contexto*, *instanciação*, *interconexão* e *ativação* de módulos. Na definição de contexto são especificados os tipos de módulos que constituirão o sistema. Na instanciação, especifica-se a criação das instâncias a partir dos tipos definidos na etapa anterior. Na fase de interconexão, descreve-se a maneira como as instâncias serão interconectadas e na ativação é disparada a execução das instâncias dos módulos.

A reconfiguração de um sistemas envolve não só a inclusão, mas também a remoção de módulos e conexões. A Linguagem de Configuração fornecer comandos que implementam as funções inversas àquelas apresentadas anteriormente, ou seja, *desconexão e remoção de instâncias de módulos*, *remoção de tipos de módulos do contexto do sistema* e *desativação de módulos*. A Linguagem de Configuração fornece independência entre as funções que representam cada uma das etapas de configuração. Isto é essencial para se obter maior flexibilidade na especificação de (re)configuração de sistemas. Um aspecto peculiar às linguagens de configuração usuais é a existência de uma ordem rígida para os comandos de configuração. Desta forma, todos os módulos são introduzidos de uma única vez, depois as instâncias são criadas e posteriormente são estabelecidas as ligações. Em **DisCo**, porém, esta rigidez não é aplicada, uma vez que restringiria a capacidade da linguagem, principalmente com relação à flexibilidade necessária para a construção de sistemas tolerantes a falhas.

A Linguagem de Configuração suporta abstração estrutural e decomposição modular por meio do uso de *módulos-configuração*. Um módulo-configuração é construído através da composição hierárquica de outros módulos. Para serem usados, os módulos-configuração precisam ser importados, exatamente como um módulo-tarefa. O objetivo de usar construtores distintos para módulos-configuração e módulos-tarefa é introduzir maior *clareza* ao processo de configuração de sistemas, descrevendo explicitamente a sua estrutura hierárquica. A principal diferença entre módulos-tarefa e módulos-configuração está na interface. A interface de módulos-tarefa é estática e definida por suas portas. Os módulos-configuração, por sua vez, são reconfiguráveis, portanto, suas interfaces são dinâmicas. Enquanto um módulo-tarefa é construído com os comandos da Linguagem de Programação, um módulo-configuração é construído apenas com os comandos da Linguagem de Configuração, através da composição de outros módulos em um módulo único.

A Linguagem de Configuração permite “escrever” expressões para especificar condições de configuração. Estas expressões, chamadas *expressões de configuração*, são formadas por predicados e primitivas que retornam informações sobre componentes de configuração, por exemplo: existem predicados para “chegar” se uma instância está ou não ativa (*IsActive*), se duas instâncias estão conectadas (*IsConnected*), etc, e primitivas que, por exemplo, retornam onde uma determinada instância ativa está executando (*Where*).

Uma característica particular da linguagem de configuração de **DisCo** é o fornecimento de mecanismos que permitem mudar a configuração de um sistema após a execução de certos

módulos, ou seja, para que logo após a execução de um módulo um outro seja ativado. Para isto, existe um comando especial WAIT (*comando de reconfiguração sincronizada*) que suspende o programa de configuração até que uma dada instância termine sua execução normalmente ou seja desativada. Esta característica é de grande importância na construção de sistemas tolerantes a falhas. Este mecanismo é geral, pois testa não apenas falhas de hardware, mas também de software. Esta característica não é encontrada em outros ambientes distribuídos como CONIC, por exemplo.

2.4 Ambiente de Execução

O ambiente de execução é constituído por um conjunto de módulos que suportam o modelo de gerenciamento de configuração dinâmica adotado em DisCo [9].

• Modelo de Gerenciamento de Configuração baseado em Conexões

Durante o processo de configuração dinâmica, pode haver um maior ou menor comprometimento do sistema, ou seja, pode ser muito ou pouco afetada a funcionalidade do sistema. Como pretendemos que o ambiente DisCo seja usado na construção de qualquer tipo de sistema, inclusive sistemas de controle em tempo real, o ideal é que o modelo de gerenciamento utilizado introduza mudanças dinamicamente afetando o mínimo possível a estrutura do sistema. Com esse objetivo, adotamos em nosso ambiente o modelo para gerenciamento de mudança de configuração baseado em conexões, que introduz modificações na estrutura de sistema dinamicamente comprometendo ao mínimo a estrutura do sistema durante o processo de reconfiguração [7, 9].

Toda e qualquer mudança de configuração afeta única e exclusivamente nós² e/ou conexões, que são as *entidades de configuração* de qualquer sistema. As entidades de configuração têm seus estados de aplicação mapeados em um conjunto de estados de configuração, de forma que o sistema de gerenciamento seja capaz de conduzir estas entidades para novos estados, onde a reconfiguração seja implementada de forma *segura e consistente*.

O estado de configuração de um nó é definido em função dos estados de configuração de suas conexões. Um nó está no estado ATIVO, quando possui ao menos uma de suas conexões ativa. Neste estado o nó opera normalmente. Quando no estado PASSIVO, o nó possui todas as suas conexões estáveis. Neste estado o nó está pronto para ser inserido ou removido do sistema.

Uma conexão no estado ATIVO opera normalmente, significando que transações podem ser iniciadas e respondidas através dela. Quando no estado ESTÁVEL, a conexão pode ser retirada do sistema com segurança. BLOQUEANDO é um estado transitório no qual uma conexão se encontra enquanto transita de ATIVO para ESTÁVEL. Neste estado uma conexão espera o término de uma transação pendente.

O protocolo de mudança é composto por um conjunto de *regras de mudança* e por um *algoritmo de mudança* que fazem com que o sistema de gerenciamento de configuração conduza as entidades afetadas pela modificação aos estados de configuração adequados à introdução consistente das modificações.

Regras de Mudança

Uma mudança de configuração está restrita à estrutura do sistema, e a sua especificação envolve apenas os comandos da Linguagem de Configuração: *create* e *delete* representando a criação

² Estamos chamando de nó um módulo em execução.

e remoção de nós, e *link* e *unlink* representando a criação e remoção de conexões. A seguir apresentamos as regras de mudança associadas com cada um destes comandos:

(i) CRIAÇÃO DE UMA CONEXÃO: a pré-condição para que um conexão seja criada é que o nó já tenha sido criado.

(ii) REMOÇÃO DE UMA CONEXÃO: a pré-condição para que um conexão seja removida é que a mesma esteja no estado estável.

(iii) CRIAÇÃO DE UM NÓ: a pré-condição para que um nó seja criado é sempre verdadeira.

(iv) REMOÇÃO DE UM NÓ: a pré-condição para que um nó seja removido é que todas as suas conexões tenham sido removidas.

Algoritmo de Mudança

Ao receber uma especificação de mudança, o sistema de gerenciamento de configuração cria, e posteriormente executa, um programa implementando a modificação especificada. Este programa, chamado de *programa de configuração*, consiste de uma seqüência de comandos gerados segundo o algoritmo de mudança apresentado a seguir.

O algoritmo de mudança é definido de acordo com as regras de mudança. À medida que o programa de configuração é executado, o sistema de aplicação é conduzido ao estado adequado e as modificações são implementadas. O algoritmo de mudança consiste dos seguintes passos:

1. Determina-se o conjunto de conexões a serem retiradas do sistema (*conjunto estável*).
2. Determina-se o conjunto de conexões a serem introduzidas no sistema (*conjunto de conexões*).
3. Determina-se o conjunto de nós a serem removidos do sistema (*conjunto de remoção*).
4. Determina-se o conjunto de nós a serem introduzidos no sistema (*conjunto de nós*).
5. Executa-se a seguinte seqüência de comandos de mudança:
 - (a) Bloqueiam-se as conexões do conjunto estável. As conexões que possuem transações pendentes são colocadas no estado temporário BLOQUEANDO. Quando as transações pendentes terminam as conexões atingem o estado ESTÁVEL.
 - (b) Neste ponto, os nós a serem removidos alcançam, automaticamente, o estado PASSIVO.
 - (c) Removem-se as conexões do conjunto estável.
 - (d) Removem-se os nós que fazem parte do conjunto de remoção.
 - (e) Criam-se os nós do conjunto de nós.
 - (f) Criam-se as conexões do conjunto de conexões.
 - (g) As novas conexões são ativadas. Conseqüentemente, os nós criados tornam-se ativos, passando a atuar normalmente como um processo.

• Suporte de Execução

O Sistema de Gerenciamento de Configuração (SGC) é constituído por três entidades: *Gerenciador de Configuração*, *Interface de Controle* e *Servidor de Nomes*.

O Gerenciador de Configuração (GC) desempenha o papel de interface entre o SGC e o usuário (operador) do sistema de aplicação. Quando o usuário especifica uma mudança de

configuração, o Gerenciador de Configuração recebe-a, checa sua validade com respeito aos tipos de nós e portas utilizados e, se a especificação for válida, o GC gera o programa de mudança que implementará a configuração especificada. Se a especificação de mudança não for válida, o programa de mudança não é gerado e, conseqüentemente, a modificação não será implementada. Neste caso, o SGC informa ao usuário que a especificação não foi validada, para que o mesmo possa corrigi-la e submetê-la novamente.

Quando um nó é construído, o programador especifica sua *interface de aplicação* como um conjunto de portas de entrada e de saída através das quais o nó se comunica com outros nós do sistema. Além da interface de aplicação, um nó possui também uma segunda interface, chamada de *interface de controle*, que tem como função servir de ligação para que o Gerenciador de Configuração envie os comandos de mudança necessários para conduzir o nó ao estado de configuração adequado à introdução das modificações.

O Servidor de Nomes (SN) exerce um papel fundamental no processo de reconfiguração dinâmica, pois é nele que o Gerenciador de Configuração obtém as informações sobre o estado de configuração dos nós envolvidos na modificação.

Ao ser criado, o nó se conecta com o SN através da sua interface de controle e se registra enviando mensagens informando suas características como: nome, sistema ao qual pertence, endereço físico, estado de configuração, etc.

Desta maneira, o Servidor de Nomes tem condições de passar as informações sobre os estados dos nós para que o Gerenciador de Configuração as utilize na implementação das modificações sobre o sistema.

3 Implementação do Ambiente

A construção do ambiente **DisCo** envolve a implementação do ambiente de programação [8, 10] (Linguagem de Programação dos Módulos e Linguagem de Configuração), a implementação dos módulos do ambiente de suporte [9] (Gerenciador de Configuração, Servidor de Nomes e Interface de Controle) assim como a implementação do protocolo de mudança [7, 9], e a construção de uma interface gráfica orientada a janelas que representa o meio de interação entre o usuário e o ambiente **DisCo**.

Foi desenvolvida uma versão inicial do ambiente onde foram implementados os módulos do ambiente de suporte, o protocolo de mudança e uma versão simplificada da interface com o usuário. O ambiente de execução foi implementado utilizando-se o ambiente OPS (OCCAM Programming System) [13] por ser o único ambiente disponível com suporte à programação concorrente, na época de sua realização. As dificuldades encontradas na implementação da primeira versão de **DisCo** se deu principalmente pelo modelo de programação estático do ambiente OPS, pelas restrições de modularidade e flexibilidade apresentadas por OCCAM [14] e pela falta recursos gráficos para a construção de interfaces. Além do ambiente de suporte, implementamos também um sistema de aplicação em cima do qual foi testada a funcionalidade do ambiente **DisCo** no processo de reconfiguração dinâmica de sistemas. Apesar das limitações, esta primeira implementação do ambiente **DisCo** foi importante, pois a partir dela tivemos uma boa idéia da complexidade associada à implementação do referido ambiente.

Com a disponibilidade do ambiente UNIX juntamente com a rede de estações de trabalho Sun, dos ambientes de programação (C, C++, Sather, etc), de ferramentas para desenvolvimento de interfaces baseadas em janelas com padrão XView, e dos utilitários `lex` e `yacc` para o desenvolvimento do pré-compilador para as linguagens do ambiente de programação, iniciamos a implementação da segunda versão do ambiente **DisCo**.

Nesta segunda versão, os módulos do ambiente de suporte estão sendo implementados em C++ [15]. A escolha de C++ se deu por dois motivos principais: compatibilidade com a biblioteca de RPC (“Remote Procedure Call”) da Sun [16], que é utilizada para implementar a comunicação entre os módulos que suportam o ambiente de execução e para chamada de rotinas do sistema operacional SunOS, e facilidades de programação introduzidas pelas características do paradigma de objetos (classes, herança, sobrecarga de operadores, etc). A interface com o usuário será formada por janelas através das quais o usuário poderá verificar o estado do sistema de aplicação como um todo, de nós do sistema, de conexões de nós, o usuário poderá ter um desenho mostrando a estrutura do sistema, ou seja, a interconexão dos diversos nós que compõem o sistema, qualquer modificação introduzida no sistema é refletida no desenho de modo que o usuário tenha sempre uma informação atualizada sobre a estrutura do sistema. A interface com o usuário terá um editor específico para construção de sistemas distribuídos, de forma que os comandos sejam colocados com sua sintaxe correta e o usuário preencha os campos com os parâmetros relativos ao sistema sendo implementado, isto é verdade tanto para a Linguagem de Programação de Módulos, quanto para a Linguagem de Configuração. À medida que o ambiente for implementado, novas características poderão ser acrescentadas.

4 Conclusão

O ambiente **DisCo** foi projetado de modo a oferecer todos os recursos necessários para construção e evolução dinâmica de sistemas distribuídos. **DisCo** adota uma metodologia de programação a nível de configuração, oferecendo linguagens distintas para programação de módulos e configuração de sistemas. Dessa forma, **DisCo** fornece *clareza* na descrição de sistemas e *flexibilidade* na sua configuração. A *modularidade* fornecida pelo ambiente de programação de **DisCo** *simplifica* o processo de programação distribuída e permite a *reusabilidade* de software. O ambiente de execução utiliza um modelo de gerenciamento de configuração dinâmica que permite reconfiguração dinâmica de forma *segura*, *consistente* e com o *comprometimento mínimo* da funcionalidade do sistema. Os recursos que o ambiente oferece através da interface gráfica com o usuário simplificam não só a construção, mas também o controle e a evolução de sistemas distribuídos, características essenciais em qualquer ambiente que visa ser empregado no desenvolvimento de aplicações reais.

Referências

- [1] Tanenbaum, A.,S.; Renesse, R.V.: “DISTRIBUTED OPERATING SYSTEMS”, *Computer Surveys*, Vol. 17, Nº 4, Dezembro 1985.
- [2] Sloman, M.; Kramer, J.; Magee, J.: “THE CONIC TOOLKIT FOR DISTRIBUTED SYSTEMS”, *Proc. of 6^o IFAC Distributed Computer Control System Workshop*, Monterey, Maio 1985.
- [3] Barbacci, M.R.; Weinstock, C.B.; Wing, J.M.: “DURRA: LANGUAGE SUPPORT FOR LARGE-GRAIN PARALLELISM”, *Parallel Processing and Applications*, pp. 371-379, 1988.
- [4] LeBlanc, T.; Friedberg, S.: “HPC: A MODEL OF STRUCTURE AND CHANGE ON A CONNECTIVITY NETWORK”, *IEEE Transaction on Computers*, C-34(12), Dezembro 1985.
- [5] Sloman, M.; Kramer, J.; Magee, J.: “CONFIGURATION SUPPORT FOR SYSTEM DESCRIPTION, CONSTRUCTION AND EVOLUTION”, *Proc. of 5th Int. Workshop on Software Specification and Design*, Pittsburgh, Maio 1989.

- [6] Kramer, J.; Magee, J.; Young, A.: "A REFINED MODEL OF CHANGE MANAGEMENT IN DISTRIBUTED SYSTEM", Department of Computing, Imperial College of Science and Technology, Agosto 1989.
- [7] Lima Filho, J.,A.; Cunha, P.,R.,F.: "MODELO BASEADO EM CONEXÕES PARA GERENCIAMENTO DE CONFIGURAÇÃO DINÂMICA DE SISTEMAS DISTRIBUÍDOS", *9º Simpósio Brasileiro de Redes de Computadores*, Santa Catarina, Maio 1991.
- [8] Justo, G.R.R.: "AMBIENTE DE PROGRAMAÇÃO DISTRIBUÍDO COM CONFIGURAÇÃO DINÂMICA DE PROCESSOS", *Tese de Mestrado*, Depto. de Informática, Universidade Federal de Pernambuco, Setembro 1988.
- [9] Lima Filho, J.,A.: "DESENVOLVIMENTO DE UM MODELO BASEADO EM CONEXÕES PARA CONFIGURAÇÃO DINÂMICA DE SISTEMAS DISTRIBUÍDOS", *Tese de Mestrado*, Depto. de Informática, Universidade Federal de Pernambuco, Março 1991.
- [10] Justo, G.R.R.; Cunha, P.R.F.: "AN ENVIRONMENT FOR DISTRIBUTED PROGRAMMING WITH DYNAMIC CONFIGURATION OF PROCESSES", *Revista Brasileira de Computação*, 5, Julho 1989.
- [11] Justo, G.R.R.; Cunha, P.R.F.: "PROGRAMMING DISTRIBUTED SYSTEMS WITH CONFIGURATION LANGUAGES", *Configurable Distributed Systems Workshop*, Março 1992.
- [13] "OCCAM PROGRAMMING SYSTEM", *STRIDE OPS User Manual*, INMOS Group of Companies, Julho 1985.
- [14] Kerridge, J.: "OCCAM PROGRAMMING: A PRACTICAL APPROACH", *Blackwell Scientific Publications*, 1987.
- [15] Stroustrup, B.: "AN OVERVIEW OF C++, *ACM Sigplan Notices*, Outubro 1986.
- [16] Corbin, J.,R.: "THE ART OF DISTRIBUTED APPLICATIONS", *Springer-Verlag*, 1991.