Um Ambiente Distribuído para Suporte à Configuração Dinâmica

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.*

1 Introdução

Os avanços tecnológicos e acadêmicos ocorridos ultimamente nas áreas de redes de computadores e sistemas distribuídos têm possibilitado não apenas o compartilhamento de recursos, mas principalmente a distribuição do processamento. Isto tem levado ao surgimento de ambientes para desenvolvimento de sistemas distribuídos. Os recursos oferecidos por tais ambientes determinam os tipos de aplicação que podem ser desenvolvidas. Os primeiros ambientes (linguagens) de programação distribuída (CSP[1], SR[2], Ada[3], etc) permitiam a construção de sistemas *estáticos*, ou seja, sistemas cuja estrutura permanece constante durante sua execução. Logicamente, este tipo de sistemas possue um espectro de aplicação limitado, uma vez que a introdução de mudanças de configuração exige a parada de sua execução. O modelo de sistemas estático não reflete a maioria dos sistemas reais existentes. Daí a necessidade de ambientes distribuídos que permitam que sistemas tenham sua estrutura alterada dinamicamente refletindo assim novas situações de execução. Assim, propriedades fundamentais de sistemas distribuídos, cómo *flexibilidade* e *robustez*, podem ser completamente exploradas para a construção de sistemas reais. CONIC[4], Durra[5] e HPC[6] são exemplos de ambientes distrbuídos já desenvolvidos com suporte à configuração dinâmica. Experiências adquiridas no contato com tais sistemas levaram ao desenvolvimento de novos ambientes que oferecem recursos não encontrados naqueles citados. Neste contexto surge o ambiente distribuído DisCo, que suporta configuração dinâmica de processos adotando uma metodologia de gerenciamento que compromete menos a estrutura do sistema e oferece recursos de programação que permitem a especificação de mudanças que possam vir a ocorrer durante a execução do sistema, o que representa uma poderosa ferramenta para tolerância a falhas.

2 Programação a Nível de Configuração

Um sistema distribuído é constituido por um conjunto de unidades de processamento, fracamente acopladas, executando em paralelo e que se comunicam através da troca de mensagens. A descrição, construção e evolução destes sistemas é simplificada separando-se a *estrutura do sistema*, formada pelos módulos que o constituem e suas interconexões, do *comportamento funcional* dos módulos individualmente[7].

Um sistema de software pode ser convenientemente descrito, construído e modificado, unicamente, em função da sua estrutura (configuração). A evolução de um sistema ocorre em função das modificações e extensões sofridas pelo mesmo. Estas modificações e extensões alteram sua estrutura (configuração) através da inclusão de novos elementos de software (processos) ou de hardware (estações), ou por meio da substituição daqueles já existentes por versões mais eficientes.

Atualmente, os ambientes distribuídos começam a utilizar a *programação a nível de configuração* [8, 9] como metodologia para desenvolvimento de sistemas, utilizando o modelo de interconexão de módulos em todas as fases do desenvolvimento de sistemas distribuídos: projeto, construção e evolução de sistemas.

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 mudanças, até mecanismos de suporte que implementem reconfiguração dinâmica de forma segura e consistente [10]. Outro foco recente de pesquisas na área de sistemas distribuídos diz respeito ao desenvolvimento de modelos específicos para gerenciar a introdução dinâmica de mudanças em sistemas

distribuídos [11, 12].

3 Ambiente DisCo

O ambiente DisCo foi concebido a partir de pesquisas realizadas pelo grupo de Redes de Computadores e Sistemas Distribuídos (ReDis) da Universidade Federal de Pernambuco (Brasil) [13, 14]. Este ambiente, atualmente em fase de implementação, adota o modelo de progamação a nível de configuração como metodologia para descrição, construção e evolução de sistemas. O modelo de gerenciamento de configuração adotado foi projetado visando o comprometimento mínimo da funcionalidade de sistemas durante o processo de reconfiguração dinâmica, característica não apresentada por modelos de reconfiguração empregados em outros ambientes [4, 5, 6]. Com relação à especificação de mudanças, DisCo oferece um comando de reconfiguração sincronizada que permite especificar antecipadamente modificações que serão implementadas em decorrência de possíveis falhas que venham a ocorrer no sistema. A seguir apresentaremos as características do ambiente DisCo. Este ambiente permite a modificação de sistemas de forma *dinâmica, segura* e *consistente*. O ambiente distribuído DisCo possui duas características principais que o diferencia de outros ambientes distribuídos existentes [4, 5, 6], 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 possíveis modificações na estrutura do sistema ainda na fase de construção do mesmo para tratar situações de falhas que sejam previsíveis.

DisCo é composto por um *ambiente de programação*, formado pelas linguagens que suportam a programação distribuída de sistemas, e por um *ambiente de execução*, formado por módulos que suportam a sua característica distribuída.

3.1 Ambiente de Programação

Um sistema distribuído é composto por um conjunto de módulos (processos) independentes interligados por meio de conexões (canais de comunicação) e que interagem entre si através da troca de mensagens para a execução de uma tarefa.

Normalmente sistemas distribuídos possuem uma estrutura complexa, 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 tais sistemas, subdividindo-o em um conjunto de tarefas menores.

Flexibilidade é uma propriedade essencial em sistemas distribuídos, porém muitas linguagens não oferecem recursos para modelar esta propriedade de forma abrangente. Com o intuito de fornecer máxima flexibilidade, optamos por definir linguagens distintas para a programação dos diversos módulos do sistema e a configuração destes módulos para refletir a estrutura global do sistema. Além de garantir *flexibilidade*,

esta solução ressalta a *modularidade* estabelecendo claramente a relação entre as diversas partes do sistema (módulos).

O DisCo possui um ambiente de programação 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. Nosso modelo de programação sofreu influências de CONIC [15, 16] devido às vantagens que esta linguagem oferece.

A Linguagem de Programação de Módulos[13, 17] é usada para descrever um *módulo-tarefa* que é uma unidade de programa sequencial[1] implementando a funcionalidade do módulo. A linguagem oferece *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 restrigem 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 portas de comunicação que constituem a interface local do módulo. Comunicação através da chamada direta a outras entidades não é permitido, pois limitaria a flexibilidade de configuração lógica de sistemas. A porta é um nome local ao módulo, desta forma todas as referências são locais. Assim, garantimos a sua *independência sintática* e asseguramos a *reusabilidade de software*. A especificação de uma interface consta na definição do sentido de comunicação das portas (entrada ou saída), os tipos de dados (informação) trocados e os tipos de transações utilizadas (notificada ou pedido-resposta).

As primitivas de comunicação 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 permitindo que um sistema seja desenvolvido e testado em uma única estação, e depois, distribuído entre as demais estações da rede.

A seguir apresentaremos um exemplo de aplicação da Linguagem de Programação na construção de um módulo servidor que recebe um pedido para execução de um serviço e envia o resultado do serviço solicitado. O recebimento do pedido e o envio da resposta são tratados pela mesma porta (service) que é do tipo pedido-resposta.

Exemplo 1:

```
TASK Module Servidor(ENTRYPORT service:t_pedido REPLAY t_servico)
     VAR pedido:t_pedido, servico:t_servico;
BEGIN
     ....
     RECEIVE pedido FROM service;
```

[1] O paralelismo ocorre a nível de módulos.

```
    ....
    SEND servico TO service;
END.
```

A Linguagem de Configuração (CL) [13, 18] é utilizada para especificar a estrutura de um sistema como um conjunto de módulos interconectados.

A especificação de um sistema distribuído consiste de 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 sistemas envolve não só a inclusão, mas também a remoção de módulos e conexões, logo, a CL 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*. 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 intâncias 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.

CL 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. 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.

CL permite "escrever" expressões para especificar condições de reconfiguração. Estas expressões, chamadas *expressões de configuração*, são formadas por predicados e primitivas que retornam informações sobre componentes de configuração, por exemplo: existem predicados para "checar" 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 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.

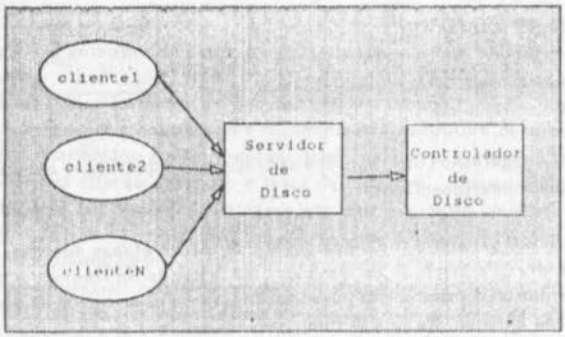

Figura 1: Sistema de Compartilhamento de Disco

A seguir ilustramos a aplicação da Linguagem de Configuração para especificar a estrutura de um sistema distribuído. Apresentamos o exemplo de um sistema que fornece serviço para gerenciar o compartilhamento de disco por múltiplos usuários. A escolha de um sistema do tipo cliente-servidor se deu principalmente por este tipo de sistema ser largamente encontrado em aplicações reais. A estrutura do sistema, chamado de *Sistema de Compartilhamento de Disco*, está ilustrada na figura 1.

O sistema de compartilhamento de disco funciona da seguinte forma: o servidor recebe os pedidos dos clientes e os encaminha para o controlador que se encarrega de controlar o acesso ao disco. No caso de falha no módulo controlador de disco é criada uma instância de um módulo gerenciador de falha que é responsável pelo diagnóstico e, se possível, pela recuperação da falha. O servidor de disco se conecta ao gerenciador de falhas e comunica aos clientes a ocorrência da falha. Se a falha for controlada, o gerente de falha encerra sua execução e o controlador de disco reinicia sua execução no sistema[2]

Exemplo 2:

```
SYSTEM Compartilhamento_Disco;
USE TASK Cliente;
USE CONFIGURATION Servidor, Controlador, Gerenciador;
BEGIN
   CREATE FAMILY k:[1..ncliente] cliente[k] FROM Cliente;
   CREATE servidor_disco FROM Servidor;
   CREATE controlador_disco FROM Controlador;
   CREATE gerenciador_falhas FROM Gerenciador;
   LINK FAMILY k:[1..ncliente] cliente[k].request TO servidor_disco.service;
   LINK servidor_disco.io TO controlador_disco.io;
   ACTIVATE FAMILY k:[1..ncliente] cliente[k];
   ACTIVATE servidor_disco, controlador_disco;
```

[2]Estamos assumindo que os módulos que compõem o Sistema de Compartilhamento de Disco já foram definidos através da Linguagem de Programação dos Módulos.

```
  REPEAT
    WAIT controlador_disco (*espera interrupção na execução do controlador (falha)*)
    ⇒ UNLINK servidor_disco.io FROM controlador_disco.io;
      LINK servidor_disco.io TO gerente_falha.io;
      ACTIVATE gerente_falha;
    END;
    WAIT gerenciador_falha (*espera gerenciador terminar execução (falha controlada)*)
    ⇒ LINK servidor_disco.io TO controlador_disco.io;
      UNLINK servidor_disco.io FROM gerente_falha.io;
      ACTIVATE controlador_disco;
    END;
  UNTIL forever;
END.
```

O modelo de programação do DisCo garante a flexibilidade necessária ao fornecimento de configuração dinâmica e a modularidade necessária para simplificar a construção e evolução de sistemas distribuídos. Todo o ambiente de programação de DisCo está definido em [13].

3.2 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 [14].

3.2.1 Modelo de Gerenciamento de Configuração

Durante o processo de configuração dinâmica, pode haver um maior ou menor comprometimento do sistema, ou seja, sua funcionalidade pode ser mais ou menos afetada. 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 [12, 14], que permite modificações dinâmicas na estrutura de sistema comprometendo ao mínimo a estrutura do mesmo durante o processo de reconfiguração. Isto é possível graças a independência existente entre o domínio da aplicação e o domínio do ambiente de configuração. Apesar desta independência, é necessário que haja uma interação entre o sistema aplicação e o sistema de gerenciamento de configuração para que o sistema de gerenciamento de configuração conduza a reconfiguração de forma adequada. Para isto, existe uma interface entre os dois domínios. Esta interface é composta por um conjunto de estados de configuração.

Toda e qualquer mudança de configuração afeta única e exclusivamente nós[1] e/ou conexões, que são as *entidades de configuração* de qualquer sistema. As entidades de configuração têm seus estados na

[1] Estamos chamando de nó um módulo em execução.

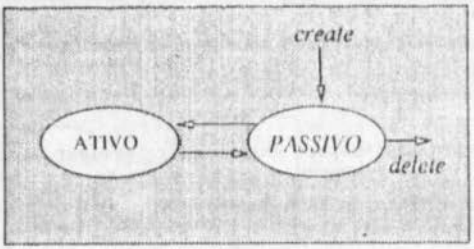

Figura 2: Diagrama de transição de estados de um nó.

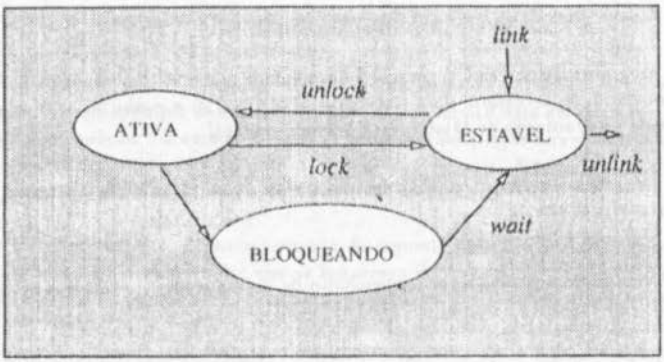

Figura 3: Diagrama de transição de estados de uma conexão.

aplicação mapeados para um conjunto de estados de configuração. Assim, o sistema de gerenciamento é capaz de conduzir as 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 gerenciamento de configuração dinâmica segue um protocolo composto por um conjunto de *regras de*

mudança e por um *algoritmo de mudança*, que faz com que o sistema de gerenciamento de configuração conduza as entidades afetadas pela modificação aos estados de configuração adequados à introdução consistente de mudanças.

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 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 *programa de configuração*, consiste de uma seqüência de comandos gerados segundo o algoritmo de mudança apresentado a seguir. Para construir o programa de configuração, o sistema de gerenciamento utiliza uma linguagem própria contendo comandos de configuração e comandos de transição. Os comandos de configuração são os mesmos da linguagem de configuração(*create*, *delete*, *link* e *unlink*). Os comandos de transição, por sua vez, são utilizados para conduzir as entidades de configuração (nós e conexões) aos estados de configuração adequados à introdução consistente de mudanças. Estes comandos são: *lock* conduz uma conexão do estado ativo para o estável, *unlock* faz o inverso do *lock* e *wait* conduz uma conexão para o estado estável após o término de uma transação pendente. Um sistema está pronto para ser modificado quando todas as conexões envolvidas na modificação atingem o estado ESTÁVEL e todos os nós envolvidos na modificação alcançam o estado PASSIVO, o que ocorre quando todas as conexões do nó estão estáveis.

O algoritmo de mudança segue as regras de mudança apresentadas anteriormente. 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 (comando *lock*). 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 (comando *unlink*).

   (d) Removem-se os nós que fazem parte do conjunto de remoção (comando *delete*).

   (e) Criam-se os nós do conjunto de nós (comando *create*).

   (f) Criam-se as conexões do conjunto de conexões (comando *link*).

   (g) As novas conexões são ativadas (comando *unlock*). Consequentemente, os nós criados tornam-se ativos, passando a atuar normalmente como um processo.

A seguir ilustramos o funcionamento do protocolo acima na implementação de uma mudança de configuração sobre o Sistema de Compatilhamento de Disco ilustrado na figura 1. Suponha que houve uma falha no módulo controlador de disco e este será retirado do sistema para a introdução do gerenciador de falhas. O modelo de gerenciamento de configuração dinâmica baseado em conexões implementaria esta reconfiguração da seguinte forma:

Exemplo 3:

PASSO 1: CONJUNTO ESTÁVEL: formado pela conexão existente entre o servidor de disco e o controlador de disco, a única conexão a ser removida nesta reconfiguração.

PASSO 2: CONJUNTO DE CONEXÕES: formado pela conexão ligando o servidor de disco ao gerenciador de falhas, única conexão a ser criada.

PASSO 3: CONJUNTO DE REMOÇÃO: formado pelo nó controlador de disco, único nó a ser removido do sistema.

PASSO 4: CONJUNTO DE NÓS: composto pelo nó gerenciador de falha, único nó a ser introduzido no sistema.

PASSO 5: Seqüência de comandos de mudança (programa de mudança):

```
manage Compartilhamento_Disco
lock     servidor_disco.io to controlador_disco.io
unlink   servidor_disco.io from controlador_disco.io
```

```
delete   controlador_disco
create   gerenciador_falha
link     servidor_disco.io to gerenciador_falha.io
unlock   servidor_disco.io to gerenciador_falha.io
```

Os comandos do programa de mudança, gerados a partir da especificação de configuração construída pelo usuário com comandos da linguagem de configuração, serão utilizados pelos módulos do ambiente de suporte de execução para implementar a modificação especificada.

3.3 Ambiente de Suporte de Execução

O ambiente de suporte contém o *Sistema de Gerenciamento de Configuração* (SGC), que é responsável pela execução do modelo de gerenciamento baseado em conexões [14]. O SGC é constituído por três módulos: *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 GC recebe-a, checa sua validade com respeito aos tipos de nós e conexões utilizados e, se a especificação for válida, o GC executa o algoritmo de mudança para gerar o programa 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 através das quais o nó se comunica com os outros nós do sistema. Além da interface de aplicação, um nó possui também uma segunda interface, que não é especificada pelo usuário, e que é fundamental para o processo de reconfiguração dinâmica. Esta interface, chamada de *interface de controle*, é inserida pelo SGC no momento que é criada uma instância do nó. A interface de controle serve de canal por onde o Gerenciador de Configuração envia os comandos de mudança necessários para conduzir os nós aos estados de configuração adequados à introdução das modificações.

O Servidor de Nomes exerce um papel fundamental no processo de reconfiguração dinâmica, ele funciona como um banco de informações sobre os diversos nós dos sistemas. Nele o Gerenciador de Configuração obtém as informações sobre o estado dos nós envolvidos na modificação.

Ao ser criados, o nó se conecta com o Servidor de Nomes através da sua interface de controle e se registra junto ao Servidor enviando uma mensagem contendo as seguintes informações:

- *nome do sistema ao qual pertence*,
- *nome*,
- *tipo*,

- *endereço físico* e
- *estado de configuração.*

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.

• Tornando Robusto o Servidor de Nomes

Devido a sua importância, o Servidor de Nomes deve ser implementado de forma robusta para que não haja possibilidade de perdas de informação, o que inviabilizaria o processo de configuração dinâmica pois o Gerenciador de Configuração não teria como identificar o estado dos nós e, conseqüentemente, não seria possível conduzi-los aos estados seguros para a introdução de modificações.

Consideramos, basicamente, duas maneiras de implementar robustez no Servidor de Nomes (SN). Uma seria adotar uma política de registro periódico dos nós junto ao Servidor de forma que após uma falha o SN tivesse como recuperar as informações sobre os estados dos nós. Esta estratégia resolveria a continuidade do processo de reconfiguração dinâmica mesmo após uma falha no SN. Entretanto, a operacionalidade desta solução implicaria em sobrecarregar a rede com a troca de mensagens entre o SN e os diversos nós do sistema aplicação.

A segunda forma seria criar mais de uma instância do SN alocando-as em estações diferentes. Com o Servidor replicado, falhando uma instância, a outra assumiria a função de fornecer informações para o Gerenciador de Configuração. A implementação de SN replicado é completamente viável através do uso de mecanismos de RPC oferecido pelo SunOS, onde a atualização dos nós chamaria um procedimento remoto de atualização em cada SN. Adotaremos está segunda estratégia como forma de implementar tolerância a falhas no Servidor de Nomes.

3.4 Metodologia de Funcionamento do Ambiente de Execução

Os módulos que compõem o ambiente de execução não atuam desordenadamente, e sim, formam um conjunto harmônico responsável pelo gerenciamento do processo de configuração dinâmica do DisCo. Esta metodologia determina o comportamento exato dos módulos do ambiente de execução durante o referido processo.

O Gerenciador de Configuração, o Servidor de Nomes e os nós do sistema de aplicação executam independentemente. Logo, instâncias destes módulos podem ser criadas, destruídas ou modificadas como um processo qualquer. A diferença entre os *nós de controle* (Gerenciador de Configuração e Servidor de Nomes) e os nós do sistema de aplicação está na forma como são criados. Por serem especiais, os nós de controle só podem ser criados, destruídos ou modificados pelo sistema operacional, jamais pelo usuário.

A metodologia de funcionamento do ambiente de execução consiste nos seguintes passos:

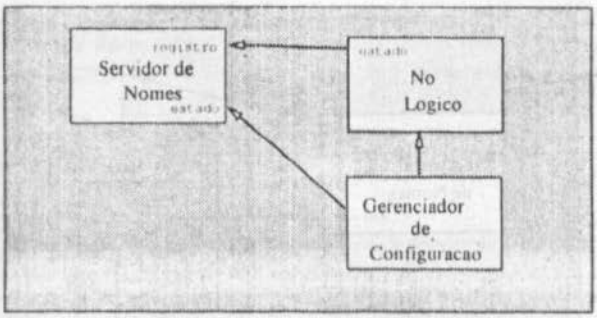

Figura 4: Módulos do suporte de execução durante uma configuração dinâmica.

- O sistema operacional cria uma instância do Gerenciador de Configuração para tratar a especificação de mudança.
- O Gerenciador de Configuração se conecta ao Servidor de Nomes e solicita os endereços dos nós envolvidos na modificação.
- O Gerenciador de Configuração se conecta com a interface de controle dos nós e valida a especificação de mudança quanto à existência e os tipos de módulos e conexões utilizados. Se a especificação for validada, o Gerenciador de Configuração gera e posteriormente envia, de acordo com o modelo baseado em conexões, uma seqüência de comandos de mudança para serem executados sobre os nós envolvidos na modificação; senão, o Gerenciador informa ao usuário que a especificação não foi validada e é, automaticamente, destruído.
- Após a execução dos comandos de mudança, os nós atingem os estados de configuração necessários à implementação segura das modificações especificadas e enviam suas mensagens de registro atualizando o banco de estados de configuração no Servidor de Nomes.
- O Gerenciador de Configuração se desconecta da interface de controle dos nós, liberando-os para novas modificações e é, posteriormente, destruído.

De acordo com esta metodologia, durante a implementação de uma configuração dinâmica os módulos que compõem o suporte de execução interconectam-se como mostrado na figura 4.

O Gerenciador de Configuração do ambiente DisCo não é distribuído, pois é composto por um módulo único, porém, instâncias do GC podem ser criadas em qualquer estação, permitindo, assim, que as operações de configuração dinâmica sejam implementadas de forma descentralizada. Isto acarreta em maior flexibilidade no processo de reconfiguração dinâmica dado que uma operação de reconfiguração pode ser iniciada a partir de qualquer estação, não ficando restrito apenas àquela onde executa a instância Gerenciador , como acontece nos ambientes que possuem um único Gerenciador de Configuração centralizado.

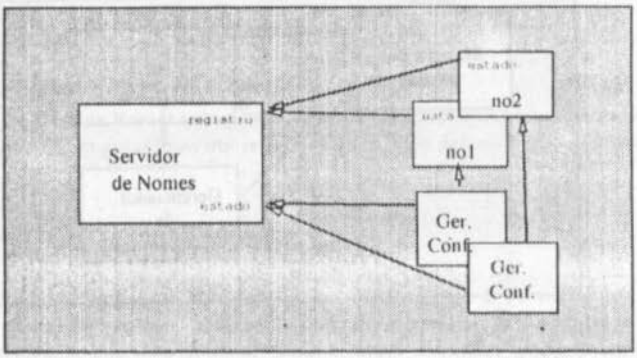

Figura 5: Módulos do suporte de execução implementando configurações simultâneas.

• **Configurações Dinâmicas Simultâneas**

O ambiente de execução suporta a implementação simultânea de mais de uma reconfiguração dinâmica, uma vez que podem existir mais de uma instância do Gerenciador de Configuração conectadas ao Servidor de Nomes, como ilustra a figura 5.

A dificuldade maior está no controle exercido pelo Servidor de Nomes para evitar a implementação simultânea de operações de configuração incoerentes sobre um mesmo sistema.

Reconfigurações dinâmicas simultâneas podem envolver sistemas distintos ou um único sistema. No primeiro caso, o controle se torna mais simples, pois é seguro que cada instância do GC implementará uma modificação sobre um sistema diferente. Daí, não há possibilidades de implementar simultaneamente modificações inconsistentes. No segundo caso, este tipo de problema pode ocorrer. Logo, o controle por parte do Servidor de Nomes(SN) na identificação dos nós que estão sofrendo modificações se torna fundamental.

No caso de modificações simultâneas sobre sistemas distintos, como existe uma única modificação, por vez, sendo implementada em cada sistema, o controle exercido pelo SN torna-se mais *simples*, uma vez que o Servidor precisa saber apenas quais sistemas estão sendo modificados naquele momento, e mais *seguro*, porque evita que modificações inconsistentes sejam implementadas simultaneamente sobre um mesmo sistema de aplicação. Quando uma instância do Gerenciador de Configuração se liga ao Servidor de Nomes e solicita o endereço dos nós envolvidos na reconfiguração, o SN "marca" aquele sistema como *sistema em configuração*, e assim, caso uma outra instância do GC tente implementar novas modificações sobre este mesmo sistema, o SN identificará que aquele sistema está sendo modificado e colocará a instância do GC que fez o segundo pedido numa fila de espera associada com o sistema.

No caso de modificações dinâmicas sobre um mesmo sistema, o controle é mais complexo, pois o Servidor de Nomes tem que saber, também, para cada sistema em configuração, quais os nós envolvidos em cada modificação. Quando uma instância do GC solicita o endereço dos nós utilizados na modificação,

o SN pesquisa se algum destes nós está sendo reconfigurado naquele momento. Em caso positivo, a instância do GC é colocada numa lista de espera associada com o referido sistema até que o(s) nó(s) seja(m) liberado(s); porém, em caso negativo, a configuração é implementada imediatamente.

4 Implementação do Ambiente

A construção do ambiente DisCo envolve a implementação do ambiente de programação [13, 17](Linguagem de Programação dos Módulos e Linguagem de Configuração), a implementação dos módulos do ambiente de suporte [14] (Gerenciador de Configuração, Servidor de Nomes e Interface de Controle) assim como a implementação do protocolo de mudança [12, 14], 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) [20] 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 [21] 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. Através de uma tela para apresentação de resultados, o usuário acompanha o processo de reconfiguração dinâmica, identifica os nós que fazem parte do sistema, identifica o estado de configuração dos nós e das conexões e verifica a saída do sistema de aplicação. A interface com o usuário, representada pela tela de apresentação de resultados, está ilustrada na figura 6.

Apesar das limitações, a implementação da primeira versão 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++, 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 progamação, iniciamos a implementação da segunda versão do ambiente DisCo.

Nesta segunda versão, os módulos do ambiente de suporte estão sendo implementados em C++ [22]. A escolha de C++ se deu por dois motivos principais: compatibilidade com a biblioteca de RPC ("Remote Procedure Call") da Sun [23], 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, e o usuário poderá ter também 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

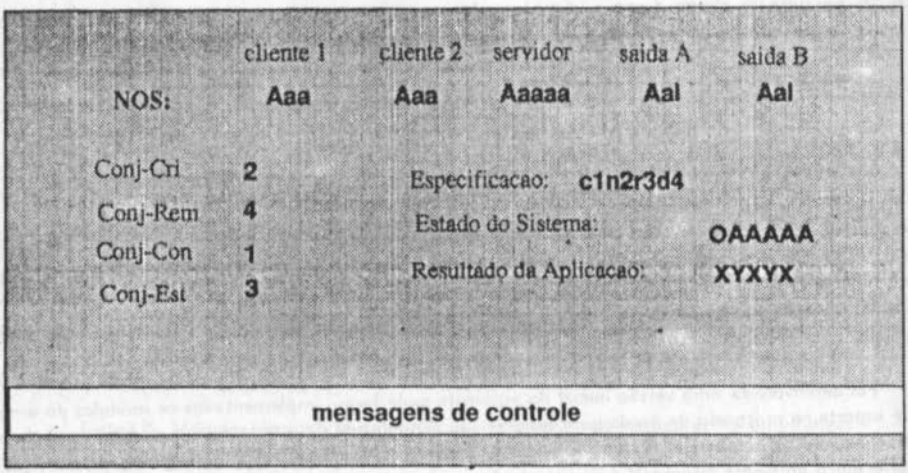

Figura 6: Tela para apresentação de resultados.

usuário terá um editor específico para construção de sistemas distribuídos, de forma que os comandos são colocados com sua sintaxe correta e o usuário preenche os campos com os parâmetros relativos ao sistema sendo implementado, isto é verdade tanto para a Linguagens de Programação de Módulos, quanto para a Linguagem de Configuração. À medida que o ambiente for implementado, novas características poderão ser incluídas.

5 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 fornecemos *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 *mínimo de comprometimento* 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, característica essencial em qualquer ambiente que se propõe a ser empregado no desenvolvimento de aplicações reais.

Referências

[1] *Hoare, C.A.R.*: " COMMUNICATING SEQUENTIAL PROCESSES", *Communications of the ACM*, Vol 21, $N^{\underline{o}}$ 8, Agosto 1978.

[2] *Andrews, G.R.*: "SYNCHRONIZING RESOURCES", *ACM Transactions on Programming Languages and Systems*, Vol 3, $N^{\underline{o}}$ 4, Outubro 1981, pp(405-430).

[3] "REFERENCE MANUAL FOR THE ADA PROGRAMMING LANGUAGE", *U.S.A. Dep. Defense*, Proposed Standard Document, Julho 1980.

[4] *Sloman, M.; Kramer, J.; Magee, J.*: "THE CONIC TOOLKIT FOR DISTRIBUTED SYSTEMS", *Proc. of* $6^{\underline{o}}$ *IFAC Distributed Computer Control System Workshop*, Monterey, Maio 1985.

[5] *Barbacci, M.R.; Weinstock, C.B.; Wing, J.M.*: "DURRA: LANGUAGE SUPORT FOR LARGE-GRAIN PARALLELISM", *Parallel Processing and Applications*, pp. 371-379, 1988.

[6] *LeBlanc, T.; Friedberg, S.*: "HPC: A MODEL OF STRUCTURE AND CHANGE ON A CONECTIVITY NETWORK", *IEEE Transaction on Computers*, C-34(12), Dezembro 1985.

[7] *DeRemer, F.; Kron, H.H.*: "PROGRAMMING IN THE LARGE VERSUS PROGRAMMING IN THE SMALL", *IEEE Transaction on Software Engineering*, Vol. SE-2, Junho 1976.

[8] *Kramer, J.*:"CONFIGURATION PROGRAMMING - A FRAMEWORK FOR THE DEVELOPMENT OF DISTRIBUTABLE SYSTEMS", *Proc. of IEEE International Conference on Computer Systems and Software Engineering (CompEuro 90)*, Israel, Maio 1990.

[9] *Finkelstein, A.; Kramer, J.; Magee, J.*: "A CONSTRUTIVE APPROACH TO THE DESIGN OF DISTRIBUTED SYSTEMS", *Proc. of* 3^{rd} *Workshop on Large Grain Parallelism, SEI/CMU*, Pittsburgh., Outubro 1989.

[10] *Sloman, M.; Kramer, J.; Magee, J.*: "CONFIGURATION SUPPORT FOR SYSTEM DESCRIPTION, CONSTRUCTION AND EVOLUTION", *Proc. of* 5^{th} *Int. Workshop on Software Specification and Design*, Pittsburgh, Maio 1989.

[11] *Kramer, J.; Magee, J.; Young, A.*: "A REFINED MODEL OF CHANGE MANAGEMENT IN DISTRIBUTED SYSTEM", Department of Computing, Imperial College of Science and Tecnology, Agosto 1989.

[12] *Lima Filho, J.,A.; Cunha, P.,R.,F.*: "MODELO BASEADO EM CONEXÕES PARA GERENCIAMENTO DE CONFIGURAÇÃO DINÂMICA DE SISTEMAS DISTRIBUÍDOS", $9^{\underline{o}}$ *Simpósio Brasileiro de Redes de Computadores*, Santa Catarina, Maio 1991.

[13] *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.

[14] *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.

[15] *Dulay, N. et al*: "THE CONIC PROGRAMMING LANGUAGE - VERSION 2.4", *Research Report DOC*, Department of Computing, Imperial College of Science and Tecnology, Outubro 1984.

[16] *Dulay, N. et al*: "THE CONIC CONFIGURATION LANGUAGE - VERSION 1.3", *Research Report DOC*, Department of Computing, Imperial College of Science and Tecnology, Novembro 1984.

[17] *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.

[18] *Justo, G.R.R.; Cunha, P.R.F.*: "PROGRAMMING DISTRIBUTED SYSTEMS WITH CONFIGURATION LANGUAGES", *Configurable Distributed Systems Workshop*, Março 1992.

[19] *Ng, K.; Kramer, J.; Magee, J.*: " GRAPHICAL CONFIGURATION PROGRAMMING", *Proc. of 22nd HICSS*, Vol II (Software Track), Havai, Janeiro 1989.

[20] "OCCAM PROGRAMMING SYSTEM", *STRIDE OPS User Manual*, INMOS Group of Companies, Julho 1985.

[21] *Kerridge, J.*: "OCCAM PROGRAMMING: A PRACTICAL APPROACH", *Blackwell Scientific Publications*, 1987.

[22] *Stroustrup, B.*: "AN OVERVIEW OF C++, *ACM Sigplan Notices*, Outubro 1986.

[23] *Corbin, J.,R.*: "THE ART OF DISTRIBUTED APPLICATIONS", *Springer-Verlag*, 1991.