Modelo Baseado em Conexões para o Gerenciamento de Configuração Dinâmica em Sistemas Distribuídos

Jorge de Araújo Lima Filho
Paulo Roberto Freire Cunha

Departamento de Informática
Universidade Federal de Pernambuco
Caixa Postal 7851, 50739 Recife - PE - Brasil
e-mail: jorge@di0001.ufpe.anpe.br

Resumo

Sistemas distribuídos desenvolvidos para aplicações críticas como controle em tempo real, utilizados em sistemas industriais, comerciais e bancários, não devem ter sua execução interrompida, seja por motivos econômicos, seja por motivos de segurança. Entretanto, tais sistemas são susceptíveis a falhas que devem ser tratadas dinamicamente, ou seja, com o sistema em execução. Os modelos para gerenciamento de configuração dinâmica existentes tratam a reconfiguração considerando um nó como única entidade de controle. Esta abordagem compromete excessivamente o funcionamento normal do sistema. Este trabalho apresenta um modelo alternativo que considera não apenas os nós, mas também suas conexões, como entidades de controle para efeito de reconfigurações. Como resultado, obtemos maior flexibilidade na introdução das modificações, afetando menos a estrutura do sistema, permitindo que uma maior parcela do mesmo continue operando normalmente durante a introdução das modificações.

Abstract

Distributed Systems for critical real-time control applications, largely used in industrial and commercial systems, can not afford being interrupted both for economic and safety reasons. However, these systems are suscetible to failures that have to be handled dynamically, i. e. , with the system still running. Current dynamic reconfiguration management models handle system reconfiguration taking into account a node as a whole, for control purposes. This approach compromises the system more than desirable during the reconfiguration process. In this article we present an alternative model which considers not only nodes, but also their interconnections in the reconfiguration control. As a result, we achieve a greater flexibility in implementing changes, reducing degradation by allowing normal operation for a larger portion of the system during reconfiguration.

1 Introdução

Modelos para gerenciamento de configuração de sistemas distribuídos são criados com o intuito de flexibilizar os ambientes para desenvolvimento de sistemas distribuídos reais. *Flexibilidade* é uma característica fundamental em sistemas distribuídos, principalmente naqueles que desempenham funções críticas, com aplicação em áreas como automação e controle industrial, sistemas bancários, comerciais e outros.

Um sistema distribuído pode ter sua estrutura modificada tanto para a redefinição de sua funcional dade, como para a recuperação de falhas. Por motivos de economia ou por questões de segurança, este tipo de sistema não deve ter sua execução completamente interrompida para que modificações sejam introduzidas em sua estrutura. Logo, faz-se necessário a existência de mecanismos que permitam a introdução de modificações de forma *dinâmica*. A principal dificuldade no fornecimento destes mecanismos é garantir que não haja perda de informações no processo de reconfiguração para que modificações sejam introduzidas de forma *segura* e *consistente*.

Um dos primeiros modelos desenvolvido para implementar configuração dinâmica utiliza uma primitiva de sincronização (**csync**) como mecanismo para evitar perdas de informações [1]. A função desta primitiva é sincronizar os nós ou processos do sistema sendo modificado fornecendo consistência ao processo de reconfiguração.

A primitiva de sincronização é introduzida no código dos processos para indicar os pontos onde estes podem ser interrompidos sem gerar inconsistências no sistema, como, por exemplo, ocorreria se um processo fosse interrompido no meio de uma transação[1]. Para que as modificações sejam introduzidas com segurança é necessário que o sistema esteja no estado de configuração adeqüado. Este estado é definido em função dos estados dos nós que o compõem.

Neste modelo, um nó possui dois estados de configuração: ATIVO ou PARADO. Um nó atinge o estado PARADO quando todos os processos que o compõem têm sua execução interrompida. Para que seja removido com segurança, é necessário inicialmente que o nó atinja o estado PARADO. Logo, é preciso que todos os processos que o compõem estejam parados; o que ocorre quando a primitiva **csync** de cada um deles é executada. Dessa maneira, é esperado que o sistema de aplicação alcance o estado ideal para a introdução de modificações de forma segura.

O problema deste modelo reside na maneira como a primitiva de sincronização é utilizada. A consistência na reconfiguração depende unicamente do programador, pois a introdução das primitivas de sincronização é tarefa exclusivamente sua. A falta da primitiva de sincronização no código de um processo implica na impossibilidade do mesmo ter sua execução suspensa. Por conseqüência, o modelo não garante consistência na introdução de modificações.

Outro modelo, chamado *modelo de configuração dinâmica baseado em nós*, construído para o ambiente distribuído CONIC [2], fornece mecanismos para tornar o processo de reconfiguração dinâmica mais seguro [3]. Neste modelo não é necessário a definição de pontos de sincronização no código dos processos, uma vez que o sincronismo é realizado automaticamente. O programador especifica a modificação desejada e, a partir daí, o próprio modelo se encarrega de implementá-la garantindo consistência e segurança. Este modelo trouxe um ganho significativo ao processo de reconfiguração dinâmica de sistemas distribuídos, uma vez que a sincronização passou a ser realizada automaticamente pelo próprio modelo, retirando do programador a responsabilidade de definir adeqüadamente os pontos de sincronização.

Os modelos citados tratam um nó como única entidade de controle no processo de reconfiguração dinâmica. Desta forma, qualquer modificação afetará um nó como um todo, mesmo que esta se restringa à modificação de uma de suas conexões. A diferença fundamental entre os modelos está na maneira como o sistema é conduzido ao estado adeqüado (seguro) para a

[1] Uma transação consiste na troca mensagens entre dois processos.

implementação de modificações. Como os modelos atuam sobre nós, podem ocorrer situações onde partes do sistema sejam afetadas *desnecessariamente*, como, por exemplo, no caso onde todas as conexões de um nó são comprometidas para que apenas uma elas seja modificada.

Por considerar de fundamental importância que a introdução de modificações na estrutura de um sistema distribuído afete o mínimo possível o seu funcionamento normal, procuramos desenvolver um modelo híbrido alternativo, mais flexível, cuja principal característica é basear o tratamento de reconfigurações tanto nos nós do sistema, quanto em suas conexões. Este modelo, denominado *Modelo Baseado em Conexões para Gerenciamento de Configuração Dinâmica em Sistemas Distribuídos*[2], constitui-se no tema deste trabalho e será apresentado na seção 5.

2 Objetivos do Modelo

O modelo baseado em conexões foi desenvolvido visando modificar sistemas distribuídos de forma *dinâmica, segura* e *consistente*. Para isto, estabelecemos os objetivos abaixo relacionados.

1. As mudanças de configuração afetam unicamente a *estrutura do sistema*. Assim, as modificações possuem maior expressabilidade e seus efeitos são mais facilmente entendidos [4]. Assumimos a utilização de ambientes de programação distribuída que forneçam linguagens distintas para programação dos nós e para configuração do sistema [5].

2. Uma especificação de mudança deve ser *declarativa*. O programador especifica apenas quais mudanças deseja e o modelo se encarrega de implementá-las.

3. Uma mudança de configuração deve deixar o sistema de aplicação em um *estado consistente*. Assim, após a modificação, o sistema de aplicação continuará sua execução normalmente não progredindo para situações inesperadas.

4. A implementação de uma mudança de configuração deve afetar o *mínimo possível* a funcionalidade do sistema. Deste modo, uma maior parcela não afetada do sistema pode continuar executando normalmente durante o processo de reconfiguração.

O último objetivo constitui-se na principal razão para desenvolvermos o modelo baseado em conexões, dado que o modelo baseado em nós, apesar de cumprir os três primeiros objetivos, não atende satisfatoriamente ao quarto objetivo especificado.

3 Modelo de Sistemas Distribuídos Adotado

Para o desenvolvimento do modelo baseado em conexões, adotamos um modelo para sistemas distribuídos segundo o qual *um sistema é constituído por um conjunto de nós interligados por conexões*. Um nó é uma entidade de processamento, que pode iniciar e responder transações com outros nós através de suas conexões, e representa não só a unidade de distribuição, como também a menor unidade de falha de um sistema distribuído. Uma conexão representa um caminho de comunicação orientado do nó que inicia uma transação para aquele que a responde. De acordo com este modelo, um sistema distribuído tem o aspecto como mostrado na figura 1.

Transação é a troca de informações entre dois nós e representa a maneira pela qual um nó pode ter seu estado alterado. Existem dois tipos de transações utilizadas em sistemas distribuídos, a saber:

[2] No decorrer deste trabalho nos referiremos a este modelo simplesmente como *modelo baseado em conexões*.

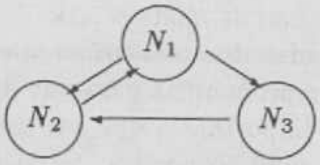

Figura 1: Representação de um sistema distribuído.

- TRANSAÇÃO PEDIDO-RESPOSTA → Representa uma transação *bidirecional*, *síncrona*, com a possibilidade de bloqueio do transmissor para a espera de respostas. O receptor também pode ficar bloqueado esperando por um pedido.

- TRANSAÇÃO NOTIFICADA → Representa uma transação *unidirecional* e *assíncrona*. Neste tipo de transação o transmissor não fica bloqueado, pois não existe resposta a esperar, no entanto, o receptor pode ficar bloqueado a espera de um pedido.

Para simplificar o desenvolvimento do modelo baseado em conexões, assumimos algumas hipóteses com relação ao modelo de sistemas distribuídos adotado. Primeiramente, foi assumido que as transações são concluídas num tempo limitado. Isto significa que toda transação termina, não existindo transações bloqueadas num estado de espera infinita. A segunda hipótese assume que o iniciador de uma transação sabe quando a mesma termina. A terceira e última hipótese assume que um sistema só pode conter transações independentes, isto é, transações cujo complemento independa de quaisquer outras transações existentes no sistema.

4 Modelo de Gerenciamento Baseado em Nós

O modelo de gerenciamento de configuração baseado em nós (modelo baseado em nós) estabelece os nós como entidades básicas para o controle de reconfigurações [3]. Isto significa que as modificações introduzidas no sistema afetarão os nós como um todo.

Este modelo assume que um sistema distribuído é composto por um conjunto de nós, que representam a sua unidade de distribuição, interligados por conexões. Logo, as modificações introduzidas em um sistema são especificadas em função da *criação ou remoção de nós* e da *criação ou remoção de conexões.*

O modelo baseado em nós separa as ações que fazem parte do *domínio de aplicação* daquelas que fazem parte do *domínio de configuração.* O domínio de aplicação é aquele onde se encontra o sistema de aplicação e o domínio de configuração é aquele que contém o Sistema de Gerenciamento de Configuração[3] (SGC), como mostra a figura 2.

A vantagem na utilização deste modelo está na simplicidade com que o programador participa do processo de reconfiguração. A tarefa do programador é "apenas" especificar a modificação. O modelo se encarrega de conduzir o sistema de aplicação ao estado adeqüado à reconfiguração, e implementá-la.

Na implementação consistente de modificações é preciso que se conheça os estados de configuração dos nós envolvidos na modificação para conduzí-los a novos estados onde as modificações sejam implementadas de forma consistente. Para isto, existe uma interface entre os domínios de aplicação e de configuração. Esta interface traduz o estado de aplicação de um nó para seu estado de configuração correspondente.

[3] O Sistema de Gerenciamento de Configuração é a parte do ambiente de execução encarregada de implementar modificações sobre os sistemas.

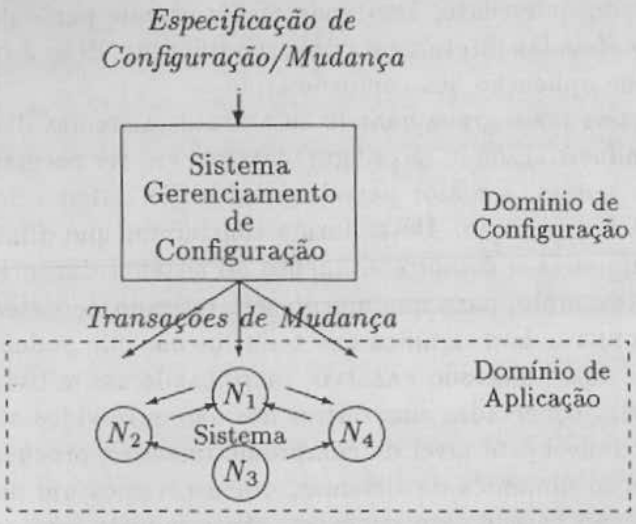

Figura 2: Domínios de Gerenciamento de Configuração.

A interface é constituída por um conjunto de estados de configuração [CRIADO, REMOVIDO, ATIVO e PASSIVO] nos quais um nó pode se encontrar. O estado PASSIVO representa o estado de configuração no qual um nó pode ser modificado. Neste estado um nó possui as seguintes características:

- as transações iniciadas pelo nó já foram concluídas,
- o nó não pode iniciar novas transações e
- o nó pode responder transações iniciadas por outros nós.

O estado passivo não é suficiente para a implementação de modificações consistentes, pois nele um nó pode responder transações iniciadas por outros nós. Por isso, é definido um novo estado no qual um nó não possua transações pendentes. Este estado é chamado *estado quiescente*, e nele um nó possui as seguintes características:

- as transações iniciadas pelo nó já terminaram,
- o nó não pode iniciar novas transações,
- o nó não está respondendo a nenhuma transação, e
- nenhum outro nó pode iniciar um transação que exija uma resposta deste nó.

A condição necessária para que um nó atinja o estado QUIESCENTE é que ele e todos os seus vizinhos diretos[4] estejam no estado PASSIVO.

O grande problema do modelo baseado em nós é que, devido as condições necessária para a alcançabilidade do estado QUIESCENTE, durante a modificação são levados ao estado passivo não apenas os nós diretamente incluídos na especificação, mas também todos aqueles necessários

[4] Um nó N é dito ser vizinho direto do nó M quando N pode iniciar uma transação com M.

para conduzí-los ao estado quiescente. Isto pode afetar grande parte da estrutura do sistema, pois envolve partes não afetadas diretamente pela modificação. Não é raro o caso em que toda a estrutura do sistema de aplicação fica comprometida.

Este problema torna-se mais grave quando se trata de sistemas distribuídos aplicados em controle e automação industrial, onde reconfigurações podem ser necessárias em decorrência de falhas, mas, ao mesmo tempo, a maior parcela possível do sistema deve continuar operando normalmente durante a recuperação. Desta forma concluímos que o modelo de gerenciamento baseado em nós não minimiza o comprometimento do sistema durante o processo de reconfiguração dinâmica. Por exemplo, para que um nó seja retirado do sistema é necessário que ele esteja no estado QUIESCENTE. Isto significa que todos os nós que podem iniciar uma transação com ele devem ser colocados no estado PASSIVO, sacrificando assim todas as conexões de saída destes nós, mesmo aquelas conectadas com outros nós não envolvidos na modificação.

Considerando inadmissível este nível de comprometimento e procurando flexibilizar mais o processo de reconfiguração dinâmica de sistemas, desenvolvemos um novo modelo para gerenciamento de reconfiguração baseado em conexões. Este modelo será apresentado na próxima seção, sendo posteriormente ilustrada a sua utilização através de um exemplo, e comparado seu desempenho em relação ao modelo baseado em nós.

5 Modelo de Gerenciamento Baseado em Conexões

Como o modelo baseado em nós, o modelo baseado em conexões foi desenvolvido de modo a tornar as ações pertencentes ao domínio de aplicação independentes daquelas de uso exclusivo no domínio de configuração (ver figura 2) [6]. Conseqüentemente, as ações necessárias para implementar qualquer mudança de configuração são determinadas e executadas exclusivamente no domínio de configuração.

A interação entre o domínio de aplicação e o domínio de configuração se dá por meio de uma interface existente entre eles. Esta interface é definida como um conjunto de estados de configuração nos quais uma conexão e um nó podem se encontrar. Os estados de configuração constituem o único meio pelo qual o modelo identifica os estados dos nós e conexões e os conduz para estados adeqüados à introdução de modificações.

Os estados de configuração de nós e conexões são apresentados a seguir, juntamente com seus respectivos diagramas de transição. Estes diagramas mostram como os nós ou conexões têm seus estados de configuração modificados de acordo com o estado original e o comando de configuração utilizado. A maneira como isto é realizado ficará mais clara quando for apresentado o algoritmo de mudança na seção 5.1.

Conjunto de Estados de Configuração de uma Conexão

- ATIVA → A conexão opera normalmente, significando que transações podem ser iniciadas e respondidas através dela.
- ESTÁVEL → A conexão existe, porém sua utilização está reservada para a inicialização e finalização de nós. Neste estado uma conexão pode ser retirado do sistema de forma segura.
- BLOQUEANDO → Estado transitório no qual uma conexão poderá se encontrar quando passa do estado ATIVA para ESTÁVEL. Neste estado uma conexão permanece até o término da transação pendente.

De acordo com os estados de configuração acima, foi definido o diagrama de transição de estados de uma conexão como ilustrado na figura 3.

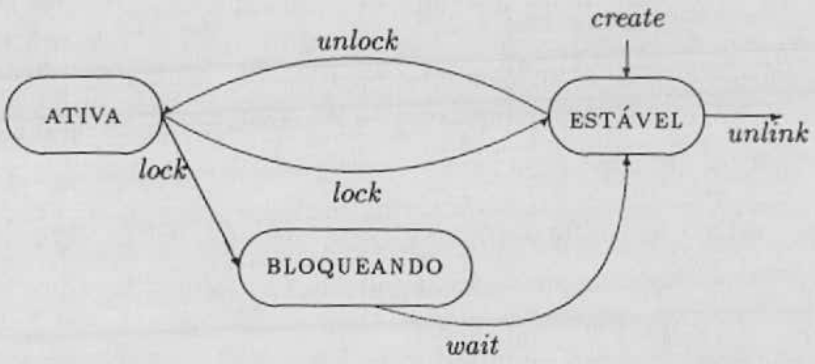

Figura 3: Diagrama de transição de estados para uma conexão.

O estado de configuração de um nó é definido em função dos estados de configuração de suas conexões, como mostrado a seguir.

Conjunto de Estados de Configuração de um Nó

- ATIVO → O nó possui ao menos uma de suas conexões ativa. Neste estado o nó opera normalmente.
- PASSIVO → O nó possui todas as suas conexões no estado estável. Neste estado o nó está pronto para ser removido do sistema.

Para os nós foi definido um diagrama de transição de estados como apresentado na figura 4.

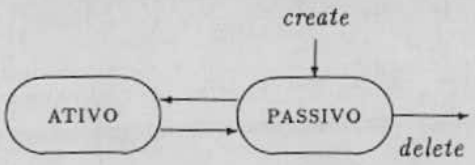

Figura 4: Diagrama de transição de estados para um nó.

Os conjuntos de estados de configuração de nós e conexões e seus respectivos diagramas de transição permitem que se implemente, de forma segura e consistente, as modificações especificadas pelo usuário. Para tanto, fez-se necessário definir um estado no qual uma conexão não possua uma transação em progresso e que, portanto, tenha estabilidade suficiente para sofrer uma modificação. Este estado é denominado de estado ESTÁVEL. Da mesma forma, o estado PASSIVO possui a estabilidade necessária para um nó ser modificado.

Uma especificação de reconfiguração escrita pelo usuário é passada ao SGC que ao recebê-la identifica o estado de configuração de cada nó e conexão contidos nesta especificação e, através dos diagramas de transição, determina a seqüência de comandos de mudança necessária para conduzir o sistema ao estado ideal para a introdução das modificações.

O usuário utiliza a linguagem de configuração do ambiente de programação distribuída para especificar a mudança de configuração desejada. Os comandos básicos desta linguagem são: **create** e **delete** para criar e remover nós e **link** e **unlink** para criar e remover conexões. A linguagem utililizada para gerar a seqüência de comandos de mudança é formada tanto pelos comandos básicos da linguagem de configuração, como também pelos comandos de transição contidos nos diagramas.

5.1 Protocolo para Gerenciamento de Mudança

O protocolo para gerenciamento de mudança (protocolo de mudança) especifica um conjunto de regras utilizadas para implementar, dinamicamente, mudanças de configuração sobre um sistema distribuído. O protocolo visa minimizar o comprometimento do funcionamento normal do sistema durante a sua modificação.

A definição do protocolo de mudança é feita em duas etapas. Na primeira são especificadas as regras de mudança que determinam as condições necessárias para a criação e remoção de nós e conexões de acordo com os comandos existentes na linguagem de configuração do ambiente de programação utilizado. Na segunda etapa é desenvolvido um algoritmo que será utilizado para gerar a seqüência de comandos de mudança (programa de configuração) responsável pela implementação consistente das modificações.

5.1.1 Regras de Mudança

Como as modificações estão restritas à estrutura do sistema é natural que a sua especificação seja descrita apenas por meio dos comandos que pertençam à linguagem de configuração do ambiente utilizado.

(*i*) CRIAÇÃO DE UMA CONEXÃO → Comando : **link**

Regra : A pré-condição para que um conexão seja criada é sempre verdadeira.

(*ii*) REMOÇÃO DE UMA CONEXÃO → Comando : **unlink**

Regra : A pré-condição para que um conexão seja removida é que a mesma esteja no estado ESTÁVEL.

(*iii*) CRIAÇÃO DE UM NÓ → Comando : **create**

Regra : A pré-condição para que um nó seja criado é sempre verdadeira.

(*iv*) REMOÇÃO DE UM NÓ → Comando : **delete**

Regra : A pré-condição para que um nó seja removido é que todas as suas conexões tenham sido removidas.

De acordo com as regras definidas anteriormente é estabelecida a seqüência de passos do algoritmo de mudança apresentado a seguir.

5.1.2 Algoritmo de Mudança

Para cada especificação de mudança é gerado um programa para implementar a mudança especificada. Este programa, chamado de *programa de configuração*, consiste em uma seqüência de comandos gerados a partir do algoritmo de mudança.

O algoritmo de mudança é definido de modo que antes da execução de cada comando de configuração as entidades envolvidas (nós e conexões) são conduzidas para estados onde a modificação pode ser introduzida de forma segura e consistente. Isto é feito segundo as regras de mudança pré-estabelecidas. Dessa forma, garantimos a estabilidade do sistema antes que os comandos de mudança sejam executados.

O algoritmo de mudança consiste dos seguintes passos:

PASSO 1: Determina-se o conjunto de conexões envolvidas na mudança de configuração, chamado *conjunto estável.*

PASSO 2: Determina-se o conjunto de nós a serem introduzidos no sistema, denominado *conjunto de criação.*

PASSO 3: Determina-se o conjunto de nós a serem removidos do sistema, denominado *conjunto de remoção.*

PASSO 4: Executa-se a seguinte seqüência de ações de mudança:

(i) – Bloqueiam-se as conexões do conjunto estável (comando *lock*). Aquelas conexões que possuem transações pendentes alcançam o estado temporário BLOQUEANDO, as que não possuem transações pendentes passam diretamente para o estado ESTÁVEL. O estado temporário é utilizado para evitar que novas transações sejam iniciadas naquelas conexões que esperam o término de transações pendentes. Quando sairem do estado temporário, as conexões atingirão o estado ESTÁVEL.

(ii) – Neste ponto, os nós a serem removidos alcançam, automaticamente, o estado PASSIVO e executam seus procedimentos de finalização, se estes existirem. A finalização consiste na última troca de mensagens entre o nó e seus vizinhos para que o mesmo deixe o sistema de forma consistente. Isto introduz um retardo na execução do programa de mudança (comando *wait*). A alcançabilidade do estado PASSIVO ocorre automaticamente a partir do momento que todas as conexões do nó alcançarem o estado ESTÁVEL.

(iii) – Removem-se as conexões do conjunto estável (comando *unlink*).

(iv) – Removem-se os nós que fazem parte do conjunto de remoção. De acordo com as regras de mudanças, os nós devem ter suas conexões já removidas (comando DELETE).

(v) – Criam-se os nós do conjunto de criação. Quando criado, o nó é colocado no estado PASSIVO (comando CREATE).

(vi) – Criam-se as novas conexões. Quando criada, uma conexão é colocada no estado ESTÁVEL (comando LINK).

(vii) – Neste ponto os nós do conjunto de criação são inicializados, pois eles estão no estado passivo e suas conexões no estado estável. A inicialização consiste na troca inicial de mensagens com seus vizinhos para que o nó entre no sistema de forma consistente. Isto introduz um retardo na execução do programa de configuração (comando *wait*).

(viii) – Após a inicialização dos nós, as conexões deixam o estado ESTÁVEL e são colocadas no estado ATIVA, onde elas podem operar normalmente (comando UNLOCK). Desta forma, os nós deixam o estado PASSIVO e atingem o estado ATIVO, onde passam a executar normalmente.

6 Aplicação do Modelo Proposto

Nesta seção será ilustrado como os modelos de gerenciamento implementam uma mudança de configuração dinâmica em um sistema distribuído. A modificação será implementada utilizando tanto o modelo baseado em nós, quanto o modelo baseado em conexões, apresentados neste trabalho.

Como os modelos atuam apenas a nível de configuração, apresentaremos a construção do sistema de aplicação a partir de um conjunto de tipos de nós que assumimos já definidos. Não nos interessa saber como cada nó foi construído, e sim, como os diversos nós são interconectados para formar o sistema de aplicação, e, principalmente, como nós e conexões são acrescentados e/ou retirados do sistema de modo a refletir as mudanças de configuração especificadas pelo usuário.

Para exemplificar adotaremos um sistema de aplicação sobre o qual será efetuada uma modificação. A modificação será implementada em cada um dos modelos e a comparação entre ambos será realizada com ênfase no nível de comprometimento que cada modelo impõe sobre a estrutura do sistema durante o processo de reconfiguração.

Utilizaremos um sistema que fornece um serviço para gerenciar o compartilhamento de periféricos (discos rígidos e impressoras) por múltiplos usuários. A estrutura do Sistema de Compartilhamento de Periféricos está ilustrada na figura 5.

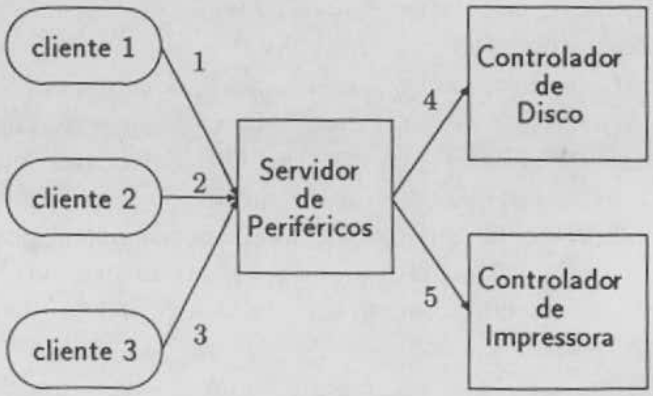

Figura 5: Sistema de compartilhamento de perifericos.

Os modelos de gerenciamento existentes foram desenvolvidos para serem utilizados em ambientes distribuídos que possuem linguagens distintas para a programação dos nós ("programming in the small") e para construção dos sistemas ("programming in the large"). Por apresentar características que fornecem flexibilidade tanto para a construção, quanto para a modificação dinâmica de sistemas, CONIC [2] foi escolhido como ambiente de programação distribuída utilizado neste exemplo.

Como foi justificado anteriormente, assumiremos já definidos, através da linguagem de programação dos módulos de CONIC [7], os tipos de nós utilizados na construção do Sistema de Compartilhamento de Periféricos. Respeitando as considerações anteriores, mostraremos como

a construção de tal sistema seria especificada utilizando-se a linguagem de configuração de CONIC [8].

```
system Sistema_Compartilhamento_Perifericos;
    use mod_usuario, mod_servidor, mod_cont_disco, mod_cont_imp;
    create family k : [1..3]
                    cliente[k] : mod_usuario;
    create servidor_de_perifericos : mod_servidor;
    create controlador_de_disco : mod_cont_disco;
    create controlador_de_impressora : mod_cont_imp;
    link family k : [1..3]
                cliente[k].pedido to servidor_de_periferico.serviço;
    link servidor_de_perifericos.IO to controlador_de_disco.IO;
    link servidor_de_perifericos.IO to controlador_de_impressora.IO;
end.
```

Vamos imaginar que, devido a uma falha, o nó Controlador de Disco deva ser retirado do sistema e substituído por um outro nó, o Gerenciador de Falha, que tratará a falha ocorrida. A nova estrutura do sistema está ilustrada na figura 6.

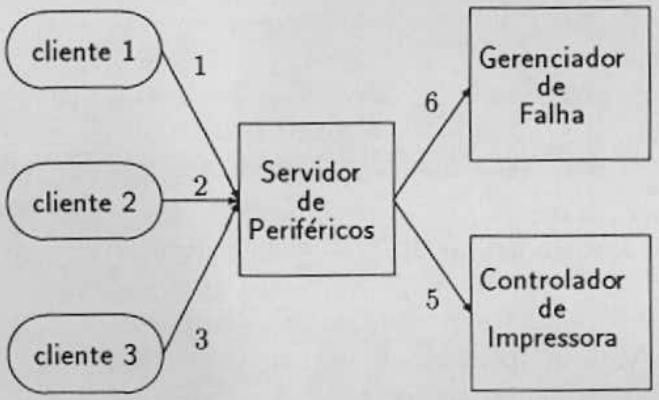

Figura 6: Sistema de compartilhamento de perifericos reconfigurado.

Essa mudança de configuração é especificada pelo usuário, através da linguagem de configuração de CONIC como mostrado a seguir.

```
change Sistema_Compartilhamento_Perifericos;
    use mod_gerenciador;
    create gerenciador_de_falha : mod_gerenciador;
    unlink servidor_de_periferico.IO from controlador_de_disco.IO;
    link servidor_de_periferico.IO to gerenciador_de_falha.IO;
    delete controlador_de_disco;
end.
```

Analisaremos, agora, como cada modelo implementaria esta modificação.

6.1 Implementação Utilizando o Modelo Baseado em Nós

O protocolo definido para este modelo [3], implementaria tal mudança de configuração da seguinte forma:

□ PASSO 1: Determinação do conjunto passivo.

Analisaremos cada comando utilizado na especificação de mudança segundo as regras definidas para este modelo.

• Comando **create**: A pré-condição para este comando é sempre verdadeira, logo, o conjunto passivo para o comando **create** é vazio.

• Comandos **link** e **unlink**: A pré-condição para os comandos **link** e **unlink** é que o nó possuidor da conexão, aquele que pode iniciar uma transação sobre a conexão, esteja no estado PASSIVO ou CRIADO. Logo, de acordo com a especificação de mudança, o conjunto passivo para estes comandos será formado pelo nó Servidor de Periféricos.

• Comandos **delete**: Para o comando **delete**, a pré-condição é que o nó a ser removido esteja no estado quiescente, implicando que o nó e todos os seus vizinhos diretos estejam no estado passivo. Logo, o conjunto passivo para o comando **delete** é composto pelos seguintes nós: Servidor de Periféricos e Controlador de Disco.

Assim, o conjunto passivo determinado no primeiro passo do algoritmo de mudança é o seguinte:

Conjunto Passivo = { Servidor de Periféricos, Controlador de Disco }

□ PASSO 2: Especificação da seqüência de comandos de configuração.

A seguinte seqüência de comandos de configuração é gerada pelo SGC a partir da especificação de mudança estabelecida pelo usuário.

manage Sistema_Compartilhamento_Perifericos
stop servidor_de_perifericos, controlador_de_disco
remove controlador_de_disco
unlink servidor_de_perifericos.IO **from** controlador_de_disco.IO
delete controlador_de_disco
create gerenciador_de_falha
link servidor_de_periferic .IO **to** gerenciador_de_falha.IO
introduce gerenciador_de_falha
start servidor_de_perifericos, gerenciador_de_falha

Analisando a seqüência de ações estabelecida, concluímos que o modelo de gerenciamento de mudança baseado em nós compromete não só o nó a ser removido (Controlador de Disco), o que é perfeitamente natural, mas também o nó Servidor de Periféricos impedindo, assim, que os clientes utilizem os serviços de impressão durante o processo de reconfiguração. Isto ocorre devido ao fato de o nó Servidor de Periféricos ter sido colocado no estado PASSIVO por causa da quiescência do nó Controlador de Disco, o que o impede de iniciar transações com o nó Controlador de Impressora.

6.2 Implementação Utilizando o Modelo Baseado em Conexões

Para implementar a mesma modificação, o modelo baseado em conexões utilizaria o protocolo definido na seção 5.1. Segundo este protocolo, a implementação envolveria os seguintes passos:

□ PASSO 1: Determinação do conjunto estável.

A determinação do conjunto de conexões que serão levadas ao estado estável, segue as regras de mudança estabelecidas no protocolo. A pré-condição para que uma conexão seja removida, comando **unlink**, é que a mesma esteja no estado ESTÁVEL. Neste caso, de acordo com a especificação de mudança do usuário, a conexão servidor_de_perifericos.IO **to** controlador_de_disco.IO fará parte do conjunto estável. Para o comando **link**, criação de uma conexão, a pré-condição é sempre verdadeira. Assim, o conjunto estável, para esse comando, é vazio.

Logo, o conjunto estável determinado no passo 1 do algoritmo de mudança é:

Conjunto Estável = { servidor_de_perifericos.IO **to** controlador_de_disco.IO }

□ PASSO 2: Determinação do conjunto de nós criados.

O conjunto de nós criados é determinado pelo comando **create**, e, neste exemplo, é formado pelo nó Gerenciador de Falha.

Conjunto de Criação = { Gerenciador de Falha }

□ PASSO 3: Determinação do conjunto de nós removidos.

O conjunto de nós removidos é estabelecido pelo comando **delete**. Logo, o conjunto de nós removidos é constituído pelo nó Controlador de Disco.

Conjunto de Remoção = { Controlador de Disco }

□ PASSO 4: Especificação da seqüência de comandos de mudança.

manage Sistema_de_Compartilhamento
lock servidor_de_perifericos.IO **to** controlador_de_disco.IO
wait
unlink servidor_de_perifericos.IO **from** controlador_de_disco.IO
delete controlador_de_disco
create gerenciador_de_falha
link servidor_de_perifericos.IO **to** gerenciador_de_falha.IO
wait
unlock servidor_de_perifericos.IO **to** gerenciador_de_falha.IO

Analisando a seqüência de comandos acima, podemos concluir que as únicas partes afetadas do Sistema de Compartilhamento de Periféricos foram o nó Controlador de Disco e uma das conexões (servidor_de_periferico.IO **to** controlador_de_disco.IO) do nó Servidor de Periféricos. Todos os demais nós e conexões do sistema não foram afetados podendo, portanto, ser utilizados normalmente durante o processo de reconfiguração. Desta forma, os pedidos de impressão solicitados pelos clientes poderiam ser atendidos normalmente, dado que a conexão servidor_de_periferico.IO **to** controlador_de_impressora.IO continua no estado ATIVA. Este exemplo ilustra como o modelo de gerenciamento de configuração baseado em conexões reduz o comprometimento da estrutura do sistema durante a implementação de reconfigurações dinâmicas. Esta redução é fruto da maior flexibilidade introduzida pelo modelo baseado em conexões em decorrência da utilização não apenas dos nós, mas também das conexões como entidades de controle no processo de reconfiguração dinâmica.

7 Considerações Finais

O modelo de gerenciamento de configuração dinâmica baseado em conexões foi desenvolvido com o intuito de minimizar o comprometimento de sistemas distribuídos, permitindo que uma parcela maior destes sistemas continue operando normalmente durante o processo de reconfiguração dinâmica. Isto ficou claro no exemplo apresentado, uma vez que na reconfiguração do Sistema de Compartilhamento de Periféricos utilizando o modelo baseado em conexões, o Servidor de Periféricos ficou livre para atender aos pedidos de impressão solicitados pelos clientes, o que não ocorreu com o modelo baseado em nós onde o Sistema de Compartilhamento de Periféricos ficou completamente comprometido, uma vez que todos os seus serviços ficaram bloqueados para a implementação da modificação especificada.

O modelo de configuração dinâmica baseado em conexões mostrou-se mais flexível que o modelo baseado em nós. Esse aumento de flexibilidade é de fundamental importância quando imaginamos a introdução de modificações em sistemas com aplicações em tempo real onde a qualquer momento a maior parcela possível do sistema deve estar funcionando normalmente sob pena de perdas econômicas ou mesmo perdas de segurança.

Devido a limitação no tamanho do trabalho, não foi possível apresentar outros exemplos ilustrativos da flexibilidade introduzida pelo modelo baseado em conexões. Entretanto, estes exemplos podem ser encontrados em [6]. Atualmente está sendo desenvolvida uma implementação com intuito de validar o modelo proposto.

Referências

[1] **Sloman, M., Kramer, J., Magee, J.**: "CONSTRUCTING DISTRIBUTED SYSTEMS IN CONIC", *IEEE Transaction on Software Engineering*, Outubro 1987.

[2] **Sloman, M., Kramer, J., Magee, J.**: "THE CONIC TOOLKIT FOR DISTRIBUTED SYSTEMS", *Proceedings of the 6th IFAC Distributed Computer Control System Workshop*, Monterey, California, USA, Maio 1985.

[3] **Magee, J., Kramer, J.**: "A MODEL FOR CHANGE MANAGEMENT", *Research Report*, Department of Computing, Imperial College of Science and Tecnology, Setembro 1988.

[4] **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.

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

[6] **Lima Filho, J.A.**: "DESENVOLVIMENTO DE UM MODELO BASEADO EM CONEXÕES PARA A CONFIGURAÇÃO DINÂMICA DE SISTEMAS DISTRIBUÍDOS", *Dissertação de Mestrado*, Departamento de Informática, Universidade Federal de Pernambuco, Março 1991.

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

[8] **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.