Um Modelo de Diagnóstico Descentralizado para Sistemas Distribuídos *

Iara de Almeida Móra [1]
Raul Fernando Weber [2]

Resumo

O objetivo deste trabalho é apresentar um modelo de diagnóstico baseado na "Teoria de Diagnóstico por 1os Princípios". Esta técnica propõem confrontar o comportamento esperado de um sistema com o comportamento observado deste; a observação é obtida a partir de testes realizados sobre o sistema. Para isto tornou-se necessário definir como o comportamento do sistema deve ser modelado, os testes necessários e determinar quais unidades do sistema vão realizar o diagnóstico.

Abstract

This article introduces a diagnostic model based on the "First Principles Theory". This technique proposes the comparison between the espected behavior from a system with its observed behavior; the observation is obtained from tests executed on the system. In order to obtain this diagnostic is necessary to define how the system behavior is modelled, which tests are needed and how to determine which units execute the diagnostic

Endereço para contato:
Universidade Federal do Rio Grande do Sul
Instituto de Informática
Curso de Pós-Graduação em Ciência da Computação (CPGCC)
Av. Bento Gonçalves 9500 Bloco IV
Caixa Postal 15064 CEP 91501 Porto Alegre - RS
FAX: (051) 336 55 76
TEL: (051) 336 83 99 Ramal 6805

* Trabalho parcialmente patrocinado pela CAPES e CNPq.

[1] Mestrando em Ciência da Computação (UFRGS); Áreas de interesse: Tolerância a Falhas, Diagnóstico de Falhas.

[2] Doutor em Informática (Karlsruhe, Alemanha, 1986), professor do Instituto de Informática e do CPGCC, UFRGS. Áreas de interesse: Tolerância a Falhas, Segurança em Sistemas de Computação, Projeto Automatizado de Sistemas Digitais

1. Introdução

Atualmente, o contínuo decréscimo do custo de processadores tem possibilitado a implantação de sistemas com um número cada vez maior de elementos processadores. Sistema Distribuído pode ser informalmente compreendido como "um conjunto de componentes autônomos que não compartilham memória e não têm acesso a um relógio comum e que fornece ao usuário uma transparência de localidade em relação aos recursos do sistema. Estes componentes estão interligados por meio de uma rede de comunicação e se comunicam unicamente por meio de mensagens enviadas através desta rede" [DRU 87].

Um sistema computacional é projetado para executar um serviço. Um serviço consiste de uma série de operações cuja execução pode ser acionada por entradas externas, feitas pelo usuário, ou pela passsagem do tempo e que produz saídas para o usuário e mudança no estado do serviço. As operações definidas por um serviço são executadas por um servidor. Um servidor executa suas tarefas sem deixar transparecer exteriormente seus estados intemediários. Para executar uma tarefa, um servidor pode dispor de serviços executados por outros servidores. Temos então, o conceito de dependência, onde um servidor *u* depende de um servidor *r* se a correção do comportamento de *u* depende da correção do comportamento de *r*. Nessa relação de dependência, o servidor *u* é chamado de cliente e o servidor *r* e é chamado de recurso.

Um Sistema Distribuído consiste de uma série de servidores em software que dependem de serviços do processador e da comunicação. Estes serviços do processador são providos concorrentemente por alguns servidores em software através do sistema operacional multiusuário. Este sistema operacional, por sua vez, depende dos serviços oferecidos pelos processadores físicos, os quais dependem dos serviços das CPU's, memórias, controladores de I/O, discos, etc. Cada um destes elementos pode falhar em algum momento, causando desde a queda no desempenho da execução da tarefa até a completa interrupção da prestação do serviço do servidor, pois por melhor que seja um dado hardware (HW), ele pode falhar.

Sempre que a confiabilidade[3] desejada de um sistema é maior que a confiabilidade dos seus componentes individuais, Tolerância a Falhas (TF) torna-se uma necessidade. Os aspectos de autonomia e independência dos componentes de um SD caracterizam uma série de vantagens potenciais em relação a TF, tais como: independência de falhas de HW, maior facilidade de isolamento de erros, maior flexibilidade de reconfiguração do sistema após falhas. Pois, como os processadores são autônomos, uma falha em um processador não afetará o funcionamento correto de outros processadores. Com a detecção da falha, o processador falho é isolado do sistema onde os outros processadores deixam de se comunicar com ele; e, dada a redundância[4] inerente de um SD, os outros processadores podem suprir os serviços do processador falho.

Para que esta reconfiguração do sistema seja possível é necessária a identificação dos processadores falhos. Este procedimento é denominado de Diagnóstico de Falhas. O Diagnóstico a nível de sistema é aplicável em sistemas cujas unidades sejam suficientemente complexas para poderem conduzir testes sobre outras unidades e o procedimento de diagnóstico pode ser invocado periodicamente quando há detecção de um erro ou ainda, antes de ser delegado um processamento crítico a uma determinada unidade.

[3] A confiabilidade é definida como sendo a capacidade que um sistema tem de responder a uma dada especificação, dentro de condições definidas e durante um certo tempo de funcionamento.

[4] Um SD possui replicações de uma aplicação, funções e dados, em diversos processadores.

2. Definições

Neste tópico serão apresentados alguns conceitos para compreendimento do Diagnóstico para Sistemas Distribuídos. Inicialmente, a definição de Tolerância a Falhas e apresentação da necessidade de se realizar o diagnóstico sobre o sistema falho. Após, para a realização do processo de diagnóstico é necessário modelar o sistema a ser diagnosticado; neste modelo há a descrição do comportamento dos componentes, ou seja, as características do sistema correto. E, para confrontar o comportamento esperado do comportamento observado do sistema, é necessário definir os componentes que vão realizar os testes e sobre quais componentes estes irão realizar.

2.1. Tolerância a Falhas

Segundo Weber [WEB 90], em alguns sistemas que falham, a frequência e o tempo de reparo são inaceitáveis ou onde o sistema se encontra é inacessível para atividades de manutenção e reparo manual. Um tratamento para estes casos é baseado em técnicas de tolerância a falhas. Um sistema pode ser projetado para ser tolerante a falhas pela incorporação de componentes adicionais e uso de algoritmos especiais, os quais tentam limitar os efeitos resultantes de falhas sobre saídas do sistema. A cobertura, ou grau de tolerância a falhas, depende do sucesso na detecção e identificação de estados errôneos correspondentes a falhas e do sucesso no tratamento destes estados e de suas causas.

Os sistemas tolerantes a falhas diferem com respeito ao seu modo de operação na presença de falhas e com relação aos tipos de falhas que devem ser toleradas: em alguns casos, o objetivo é continuar a proporcionar o desempenho e a capacidade funcional total do sistema: noutros, o desempenho degradado ou a capacidade funcional reduzida são aceitáveis, até a remoção da falha.

As atividades relacionadas à tolerância a falhas podem ser sub-divididas em quatro fases: detecção de erros, confinamento e avaliação de danos, recuperação de erros e tratamento de falhas. Quando consideradas em conjunto, constituem-se nas etapas através das quais a Tolerância a Falhas é implementada, evitando que falhas conduzam o sistema a uma situação de funcionamento inadequado.

A detecção de erros consiste no reconhecimento da existência de um estado errôneo, o qual é efeito ou resultante da manifestação de uma falha. Este estado errôneo consiste em um estado diverso previsto na especificação inicial do sistema. Assim, a detecção é o ponto de partida para as técnicas de Tolerância a Falhas. Esta fase de detecção de erros pode ser processada em diversos momentos, podendo ser concorrente com a operação normal do sistema ou ser ativada conforme a necessidade.

O tempo que transcorre desde a ocorrência de uma falha até sua manifestação através de um erro (detectável) é chamado de tempo de latência ou latência de erro. Este tempo é explicado pelo fato de que apenas a excitação da área afetada com determinado valor ou combinação de valores lógicos resultará em erro. Por outro lado, dependendo do nível onde se dá esta detecção, é possível que informações inválidas já estejam espalhadas pelo sistema, e o estado geral desse seja passível de suspeita. Assim, antes de efetuar os procedimentos previstos para o tratamento do erro detectado, faz-se necessário verificar a extensão dos danos provocados sobre o restante do sistema. Esta fase é denominada de confinamento e avaliação de danos.

Já a fase de recuperação de erro visa transformar o estado atual incorreto do sistema em um estado livre de falhas, escolhido e definido durante a especificação, a partir do qual pode ser retomada a operação normal do sistema. Finalmente, se não for identificado a unidade ou componente falho, é deixada uma abertura para que se repita o problema anterior a qualquer

momento. Portanto, é necessário providenciar mecanismos que diagnostiquem a origem do erro e atuem na correção de suas causas. Deste modo, o primeiro aspecto do tratamento de falhas consiste em tentar localizar a falha de forma precisa. Os objetivos seguintes referem-se a efetuar o reparo da falha e/ou a reconfigurar o restante do sistema para evitar a falha.

2.2. Diagnóstico de Falhas

O processo de diagnóstico consiste em descobrir as diferenças entre o modelo ideal e a realidade. Para detectar as diferenças existentes, o diagnóstico baseia-se em parâmetros, como: a estrutura do sistema, o modelo de cada componente, o conjunto de possíveis diferenças/conflitos entre o modelo e o componente real; e o conjunto de medições - resultados de testes efetuados sobre o sistema real. Este último, por sua vez, produz um conjunto de candidatos, cada um sendo um conjunto de diferenças que interpretam as observações.

As diferenças entre o componente e o seu modelo não são fruto da observação direta, devem ser inferidas, através de testes, do comportamento do componente em relação ao modelo. Neste caso, o diagnóstico é um processo incremental, isto é, a medida que o sistema realiza o diagnóstico, o espaço de candidatos vai sendo refinado. O refinamento é usado para melhor direcionar os próximos testes. Um candidato é uma hipótese particular de como o objeto atual difere do modelo. Um candidato é representado por um conjunto de suposições todas assumidas como verdadeiras.

A localização de uma falha consiste em uma descrição das diferenças entre o comportamento esperado e o comportamento observado. Esta descrição das diferenças ajuda a selecionar a tática de localização da falha. A tática de localização é um método utilizado para concentrar a atenção em pontos "suspeitos", gerando um conjunto de hipóteses de falha.

Cada hipótese de falha é então testada para determinar se o defeito esperado está presente. Se uma hipótese é verificada, então ela e o ponto que está incorreto podem ser exibidos. Caso uma hipótese falhe, as táticas de localização são reavaliadas e uma nova hipótese pode ser gerada a partir da mesma tática, ou uma nova tática é selecionada.

Resumindo, baseado no comportamento conhecido do sistema e na detecção de um comportamento anormal, são determinados como candidatos os componentes que podem ter causado este comportamento anormal, a partir deste conjunto são feitos testes e minimizações no conjunto para determinar qual o componente que realmente causou a falha.

2.3. Modelagem do Sistema

Redes Semânticas permitem representar o conhecimento através de modelos. Estes modelos podem ser formulados como grafos, com nodos representando objetos e/ou conceitos e os arcos representando suas relações. Neste tipo de organização é possível testar facilmente se as informações a respeito de um indivíduo conhecido são novas, redundantes, inconsistentes ou deriváveis a partir de relações prévias. Esta forma de representação torna-se ainda mais interessante se forem examinados os conceitos de herança, "demons" e "default". Os atributos ou características de um objeto ou classe podem conter informações estáticas, isto é, valores definidos, ou ainda procedimentos de resolução sobre os dados representados.

Numa rede semântica existem arcos especiais que transmitem a herança dos atributos na rede. A herança permite o movimento de descrições (atributos) de classes para objetos. Os objetos podem herdar os atributos de sua classe; esta, por sua vez, pode ser uma sub-classe e então herdar os atributos da super-classe, o que resulta no fato do objeto herdar o conjunto de atributos de todas as suas super-classes. Para inibir um atributo que seria herdado, é necessário

associar ao nodo um novo atributo, "quebrando a herança". A relação que faz a ligação entre o nodo e o seu atributo particular deve ser a mesma, ou seja, ter o mesmo nome.

Muitas vezes certos atributos de uma classe ou objeto não possuem valor, pois este depende da informação contida na estrutura. Assim, faz-se necessário um recurso de computação sobre a estrutura chamado de "demon". Este recurso ativa um procedimento, através dos "demons":

- "se-buscado" : quando for feita uma referência a característica;
- "se_necessário" : no momento da atribuição do valor é executado um procedimento;
- "se_especificado" : no momento da atribuição do valor é feita uma consistência.

Da mesma forma podem ser especificados valores "default" para um atributo; se o atributo não tiver valor ou um "demon" para o cálculo do valor, pode-se buscar o valor "default". A figura 2.1 procura apresentar estes conceitos.

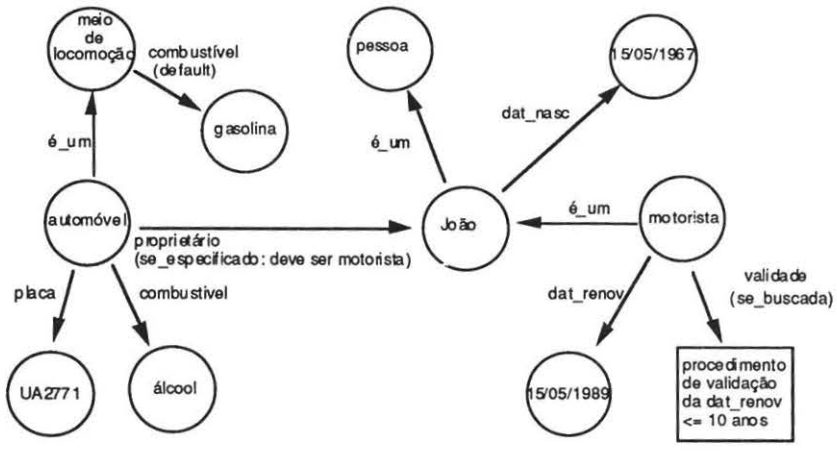

Figura 2.1 - Modelagem por Rede Semântica

Com seu crescimento, as redes semânticas podem gerar estruturas confusas. Para isto, surgiu a noção de uma estrutura de rede agregada chamada de "frame", onde as ligações associadas com um objeto particular são chamadas de "slots" (atributos).

Em um frame os níveis superiores são fixos e representam informações, que são sempre verdadeiras, sobre a situação que se está representando. Os níveis inferiores são constituídos por terminais, que podem ser preenchidos pelos nodos.

Os "slots" incluem também os tipos de relacionamento entre os diversos "frames", podendo ser relação de sub-classe/super-classe, objeto/classe, objeto/conjunto, ou ainda, subclasse/conjunto. Os "slots" podem ser preenchidos de várias maneiras: dando-se o valor, ou, se necessário relacionando um procedimento para computar o valor.

Os "frames", quando relacionados, agrupam-se para formar sistemas. A aglomeração do conhecimento forma os "scripts" que descrevem certas relações. Assim, um "frame" é uma

coleção de nodos de redes semânticas e de nodos vazios, os quais descrevem um objeto, um conceito ou uma situação.

Considerando a rede semântica apresentada, temos a representação em frames como a ilustrada na figura 2.2.

```
pessoa:
   dat_nasc:
meio_de _locomoção:
   combustível: default gasolina
automóvel:
   é_um: meio_de_locomoção
   placa:
   proprietário: se_especificado (demon: deve ser motorista)
motorista:
   é_um:
   dat_renov:
   validade: se_buscada (demon: validar data_renov (<=10 anos))
João:
   é_um: pessoa
   dat_nasc: 15/05/1967
automóvel:
   é_um: meio_de_locomoção
   placa: UA2771
   proprietário:João
   combustível : álcool
motorista:
   é_um: João
   dat_renov: 15/05/1989
```

Figura 2.2 - Modelagem por "Frames"

2.4. Modelagem do Diagnóstico

Pode-se fazer diagnóstico de forma centralizada, onde uma unidade do sistema fica responsável pelo diagnóstico do sistema; ou de forma descentralizada, um grupo de unidades faz o diagnóstico conjuntamente, onde as unidades realizam diagnósticos individuais sobre o sistema e depois chegam a um consenso sobre qual é o diagnóstico final do sistema.

O diagnóstico centralizado, para sistemas distribuídos, exige que a unidade que realiza o diagnóstico seja tolerante a falhas; pois se a unidade falhar não há mais como realizar o diagnóstico do sistema. Para uma unidade ser tolerante a falhas é necessário incorporar componentes adicionais e utilizar algoritmos especiais, o que encarece o custo do sistema.

Na forma descentralizada existem diversos modelos: a forma mais simples é cada unidade testando outra unidade; porém o sistema, como um todo, não consegue chegar a um resultado confiável de quais unidades estão boas e quais não estão. Pois, conforme a figura 2.3, tem-se:

- unidade Nome1 testa unidade Nome2 e diz que ela está boa;
- unidade Nome2 testa unidade Nome3 e diz que ela está boa;
- unidade Nome3 testa unidade Nome4 e diz que ela está **falha**;
- unidade Nome4 testa unidade Nome1 e diz que ela está boa;

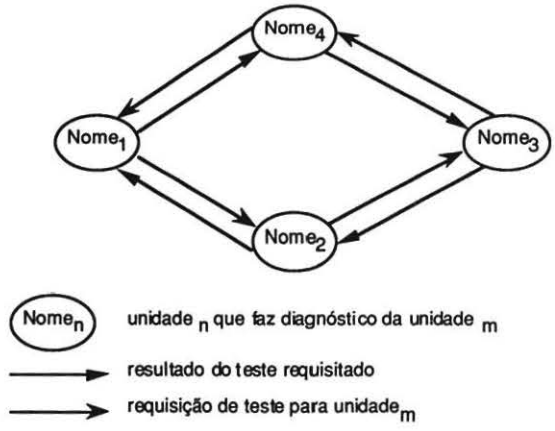

Figura 2.3 - Contra-Exemplo

Como a unidade Nome4 foi considerada falha, não pode-se confiar no que ela assumiu: que a unidade Nome1 está boa. Não podemos então confiar no que a unidade Nome1 diz sobre a unidade Nome2, e assim por diante. Este modelo não consegue chegar a um diagnóstico pois o número de testes sobre cada unidade é pequeno. Uma solução para este problema é ter mais de uma unidade requisitando testes e fazendo o diagnóstico da unidade a ser verificada. Ao final, as unidades diagnosticadoras conversam e chegam a um consenso sobre a unidade testada. O modelo ideal é ter todas as unidades testando todas as unidades, apresentado na figura 2.4; porém, como o número de unidades em um sistema distribuído pode ser elevado, torna-se custoso este modelo, pois há um acréscimo considerado de processamento e troca de mensagens.

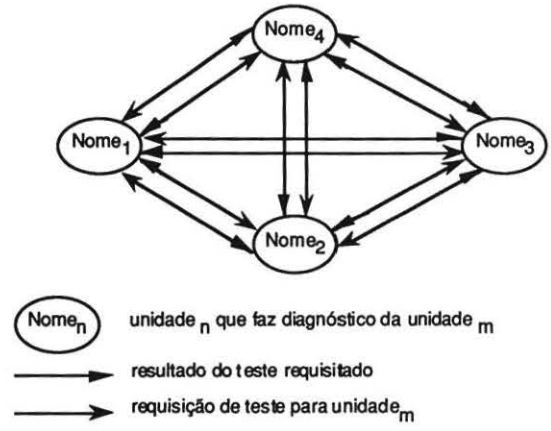

Figura 2.4 - Modelo Ideal

Uma solução para este problema é as unidades enxergarem apenas um grupo de unidades do sistema, denominado grupo_teste; o grupo que faz o diagnóstico é então denominado de grupo_diagnóstico. Cada unidade do grupo_diagnóstico requisita testes para todas as unidades do grupo_teste, realizando então diagnóstico sobre cada unidade. Após, todas as unidades do grupo_diagnóstico chegarem a seus resultados, é realizado um consenso sobre cada unidade testada. Ao final, quando todas as unidades do grupo_diagnóstico chegarem a um consenso, é enviado o resultado por multicasting[5] para o resto do sistema.

Este modelo facilita no caso de inclusões de uma nova unidade: a unidade é incluída em um grupo e esta passa automaticamente a ser enxergada pelo grupo_diagnóstico responsável, e passa a enxergar o grupo_teste. A inclusão de mais de uma unidade pode vir a gerar um novo grupo; este critério depende da técnica de difusão confiável [DUE 92] utilizada para realizar o consenso, onde o tamanho do grupo determina o menor custo em relação ao processamento e o número de mensagens trocadas.

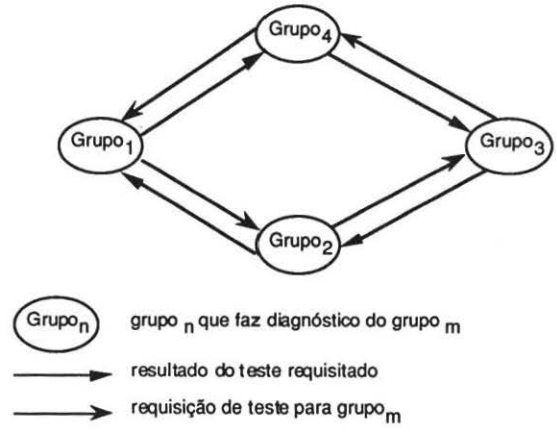

Figura 2.5 - Modelo Proposto

3. Diagnóstico Proposto

O objetivo deste trabalho é definir um sistema de diagnóstico de falhas que possa ser utilizado em um ambiente de estações distribuídas, conectadas em rede. Este modelo se baseia na "Teoria por 1os princípios" [REI 87] que apresenta uma teoria geral de diagnóstico, que propõem confrontar o comportamento esperado de um sistema com o comportamento observado deste. O resultado desta comparação é o conjunto dos componentes do sistema que causaram o comportamento anormal, ou seja, que causaram a diferença entre o comportamento esperado do comportamento observado. Para isto é necessário que estejam disponíveis:

- comportamento esperado: descrição, definida em alguma lógica, de como o sistema se comporta, assumindo que todos os componentes funcionam normalmente;

[5] É um serviço responsável por garantir, mesmo na presença de falhas, um comportamento bem definido em relação a mensagens definidas no sistema. Subentende-se aqui como difusão de uma mensagem o envio desta a um grupo de processos.

- componentes do sistema;
- comportamento observado: são os resultados de testes efetuados sobre o sistema.

A seguir será apresentado a modelagem do sistema, o de diagnóstico e a descrição das fases necessárias para a realização do diagnóstico.

3.1. Modelagem do Sistema

Neste tópico procura-se apresentar como as unidades do sistema serão modeladas. A representação escolhida baseia-se em "frames" onde o nível superior é a definição de estação, que é composta de periféricos ("slots"). Cada tipo de estação do sistema distribuído deve ser modelado, ou seja, deve ser fornecido todos os periféricos que podem pertencer aquele tipo de estação. O uso do "default" *não_tem* como atributo é para assumí-lo, por herança, caso não seja definido o periférico para aquela estação. O uso do "se_especificado" para alguns periféricos define os modelos possíveis para aquele determinado periférico:

Estação
CPU:
memoria:
disco: default não_tem
interface:
video:
teclado:
fita: default não_tem
impressora: default não_tem
scanner: default não_tem
modem: default não_tem
audio: default não_tem

Estação$_n$
é_um: Estação
CPU:
Memória: se_especificado:(demon: estar entre as opções)
Video: se_especificado:(demon: estar entre as opções)
Teclado: se_especificado:(demon: estar entre as opções)
outros...

Cada modelo de periférico por sua vez, é definido em um "frame" onde cada "slot" deste determina o teste a ser realizado no momento do diagnóstico - através dos demons: "se_necessário", "se_buscado" e "se_especificado". O "slot" *operador* em alguns periféricos faz referência a necessidade da presença de um operador para que o teste seja realizado. Por exemplo, no periférico impressora, é necessária a presença de um operador para verificar se a impressão está correta e fornecer o resultado para o sistema de diagnóstico. A seguir, a modelagem de cada periférico:

- Para o teste da Unidade Central de Processamento (CPU) se executa um programa que, a partir de parâmetros fornecidos, deve devolver resultado conhecido. Este programa deve ser genérico o suficiente para testar todas as funções da CPU:

CPU_n
Programa: se_necessario:(demon: resultado = prog(parametros))

- Para a memória são determinados testes de paridade, códigos de detecção e correção de erros (ECC), leitura e escrita na memória e endereçamento. Estes testes visam verificar a integridade

da memória: se a paridade está correta; se a execução do código de ECC é satisfatória, ou seja, se na detecção de um erro este foi recuperado; se consegue-se ler a memória sequencialmente; se com a escrita em uma posição de memória consegue-se ler o valor escrito; e se a partir de um endereço possa-se ler a posição de memória. Os testes para o disco e a fita são semelhantes aos realizados na memória.

Memória$_n$
Paridade: se_necessário:(demon: resultado = calculo_paridade)
ECC: se_necessário: (demon: resultado positivo do ECC)
Leitura: se_necessário: (demon: ler posição de memória)
Escreve: se_necessário: (demon: deve-se ler o que foi escrito)
Endereçamento: se_necessário: (demon: ler a posição de memória)

Disco$_n$
Paridade: se_necessário:(demon: resultado = calculo_paridade)
ECC: se_necessário: (demon: resultado positivo do ECC)
Leitura_seq: se_necessário: (demon: ler sequência de trilhas)
Leitura_dir: se_necessário: (demon: ler trilha)
Escreve: se_necessário: (demon: deve-se ler o que foi escrito)

Fita$_n$
Operador: se_necessário: (demon: operador disponível)
Paridade: se_necessário:(demon: resultado = calculo_paridade)
ECC: se_necessário: (demon: resultado positivo do ECC)
Leitura: se_necessário: (demon: ler memória)
Escreve: se_necessário: (demon: deve-se ler o que foi escrito)

- Para se testar a interface de comunicação, deve-se enviar uma mensagem para outros nodos da rede, via Broadcast, e estes devem re-enviar a mensagem.

Interface$_n$
Broadcast: se_necessário: (demon: enviar e receber a mensagen)

- O teste realizado no vídeo é geração de imagens, que deve ser confirmado pelo operador.

Vídeo$_n$
Operador: se_necessário: (demon: operador disponível)
Imagem: se_necessário: (demon: gerar imagem)

- Já o teste realizado no teclado é o operador digitar determinados caracteres ou combinações de caracteres determinados.

Teclado$_n$
Operador: se_necessário: (demon: operador disponível)
Caracteres: se_necessário: (demon: digitar conjunto de caracteres)

- Para a impressora, é enviado um arquivo que possibilite testar todos os recursos disponíveis, como: caracteres especiais, tipos, cores, gráficos, etc. Sendo que a impressão deve ser confirmada pelo operador.

Impressora$_n$
Operador: se_necessário: (demon: operador disponível)
Listagem: se_necessário: (demon: imprimir arquivo_teste)

- O teste realizado no scanner é, a partir de uma imagem, a confirmação do operador da cópia da imagem correta.

Scanner$_n$
Operador: se_necessário: (demon: operador disponível)
Imagem: se_necessário: (demon: copiar imagem)

- Para o teste do modem são utilizados os recursos que o próprio fornece: testes de loopback.

Modem$_n$
Operador: se_necessário: (demon: operador disponível)
Loopback: se_necessário: (demon: enviar e receber sinal (*a-b,a e b, a ou b*))

- Finalmente, o teste de áudio é a geração de som, com a confirmção do operador.

Audio$_n$
Operador: se_necessário: (demon: operador disponível)
Som: se_necessário: (demon: gerar som)

A partir da definição dos tipos de estações existentes, será modelada cada estação que pertence ao sistema, se fazendo referência do modelo de estação existente e dos periféricos; também são acrescentados os "slots" *grupo_diagnóstico* e *grupo_teste* para determinar o grupo que a estação pertence e o grupo que a estação diagnostica, respectivamente.

Nome$_n$
é_um: Estação$_n$
CPU:
Memória:
Video:
Teclado:
outros...
grupo_diagnóstico:
grupo_teste:

A figura 3.1 apresenta um exemplo de modelagem do sistema. Salienta-se que podem existir tipos de estações, sem haver este tipo no sistema. Porém não pode-se ter estações do sistema sem um tipo pré-definido.

3.2. Diagnóstico por 1os Princípios

O algoritmo gerado a partir desta teoria independe da lógica utilizada para representar o sistema pois um provador de teoremas é que, a partir da descrição do sistema, os componentes e o comportamento observado, gera um conjunto dos prováveis componentes falhos. A partir destes, o algoritmo de diagnóstico irá determinar os falhos. Com isto, para um determinado sistema de diagnóstico, pode-se ter diferentes provadores de teoremas específicos para diferentes aplicações. A seguir, as definições necessárias para a teoria de diagnóstico.

Assumindo-se que sistema é definido a partir da descrição do sistema (DS) - um conjunto finito de sentenças definidas em uma lógica - e os seus componentes (componentes) - um conjunto finito de constantes; que observações do sistema (OBS) também são descritas como um conjunto finito de sentenças em uma lógica; e que um componente anormal é representado na forma de um predicado unário: AB(.).

Diz-se que um sistema, com descrição do sistema DS e componentes Ci,...,Cn, está funcionando normalmente se DS U {not AB(Ci), ..., not AB(Cn)} for consistente, ou seja,

a descrição do sistema - que assume que todos os componentes funcionam normalmente - e o conjunto de predicados - que dizem que os componentes não estão falhos - é verdadeira. Para determinadas observações OBS que tornem a expressão DS U {not AB(Ci), ..., not AB(Cn)} U OBS inconsistente, indicando que existem componentes falhos.

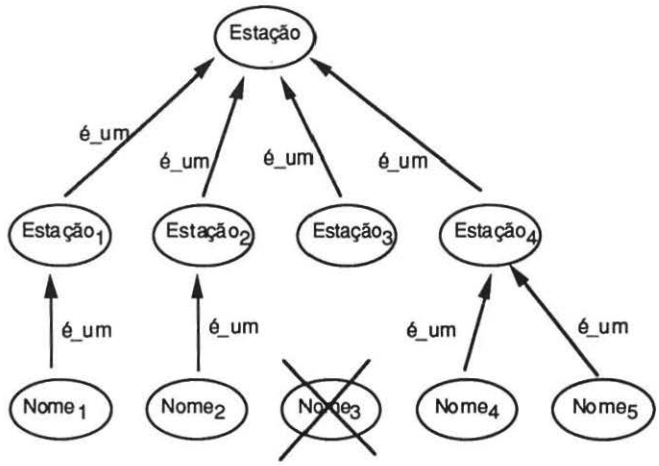

Figura 3.1 - Modelagem do Sistema

Temos que diagnóstico para (DS, componentes, OBS) é um conjunto mínimo Diag de elementos contidos no conjunto componentes de maneira que DS U OBS U {AB(c)/ c ∈ DIAG} U {not AB(c)/ c ∈ componentes - DIAG} é consistente, ou seja, nesta expressão tem-se definido o conjunto dos componentes falhos e o conjunto dos componentes não-falhos, tornando a expressão verdadeira. Simplificando a expressão acima, pode-se assumir que, com a descrição do sistema, o conjunto dos componentes, as observações do comportamento real do sistema e o conjunto dos componentes não-falhos chega-se aos componentes falhos.

Podemos definir que um conjunto de conflito (CF) é um conjunto {C1,...,Ck} contido no conjunto componentes que torna a expressão DS U OBS U {not AB(C1), ... , not AB(Ck)} inconsistente. Definimos também que este conjunto é mínimo sse nenhum dos seus subconjuntos próprios for um conjunto de conflito. Por exemplo: componentes = {1,2,3,4} e se {1,3,4} e {1,3} são conjuntos de conflito, temos somente {1,3} como conjunto de conflito mínimo pois, com os componentes 1 e 3 a falha já ocorre e o componente 4 pode ser consequência da falha.

Com estas novas definições podemos dizer que diagnóstico é um conjunto Diag para (DS,componentes,OBS) sse Diag é um conjunto mínimo tal que componentes - Diag (ou seja, o conjunto dos componentes não-falhos) não é um conjunto de conflito para (DS,componentes,OBS).

3.3. Fases do Diagnóstico

Neste tópico serão apresentadas as fases do processo de diagnóstico. Salienta-se que este processo pode ser ativado com a detecção de um erro ou pela ativação de um

processamento crítico para verificar se o sistema está em condições de processar. A primeira fase do Diagnóstico consiste em cada grupo_diagnóstico requisitar testes para cada unidade do grupo_teste. Nesta fase, temos somente troca de mensagens, onde cada grupo_diagnóstico envia uma mensagem para cada unidade do seu grupo_teste,como pode ser visto na figura 3.2.

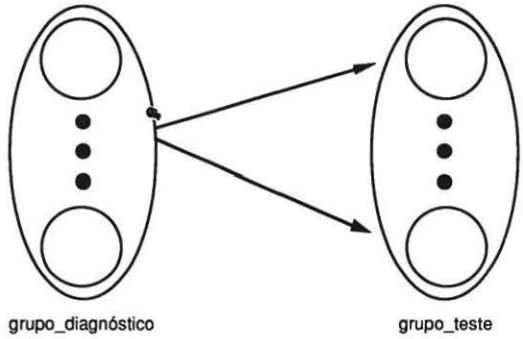

Figura 3.2 - Fase I: requisição de testes

Na segunda fase, cada unidade realiza os testes definidos na modelagem para cada periférico que o compõem; nesta fase há processamento dos testes requisitados, onde cada unidade do grupo_teste realiza os testes definidos na modelagem. Ao término dos testes, cada unidade envia para cada unidade do grupo_diagnóstico os resultados obtidos. Nesta fase, terceira, há somente envio de mensagens, como é ilustrado na figura 3.3. Na sequência, quarta fase, há processamento local do algoritmo acima referido, onde cada unidade_diagnosticadora realiza o diagnóstico de cada unidade do seu grupo_teste.

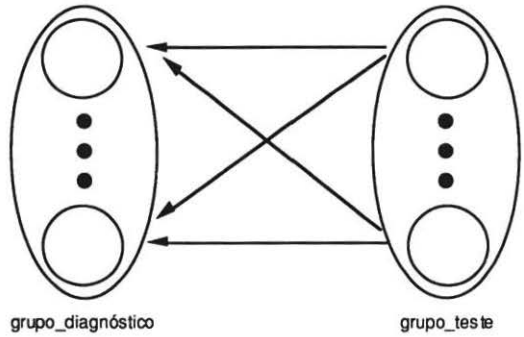

Figura 3.3 - Fase III: resultado dos testes

Após realizado o diagnóstico individual, as unidades pertencentes ao grupo_diagnóstico trocam mensagens sobre os resultados obtidos - através de um protocolo de difusão confiável - e chegam a um consenso sobre as unidades do diagnóstico do grupo_teste. Nesta fase, quinta, mostrada na figura 3.4, o número de mensagens trocadas varia conforme o protocolo utilizado, no caso da utilização do algoritmo dos Generais Bizantinos sem assinatura [LAM 82], o número de mensagens, para cada grupo é $(n(n-1) + n(n-1)(n-2) + ... + n(n-1)...(n-m)$, onde n corresponde ao número de componentes do grupo e m ao número de componentes falhos

suportados pelo grupo. Para diminuir o volume de mensagens poderia se utilizar o algoritmo dos Generais Bizantinos com assinatura, porém com um acréscimo de processamento ao sistema, pois há a necessidade de codificação e decodificação da assinatura.

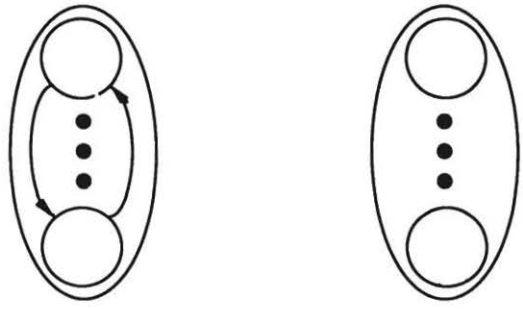

Figura 3.4 - Fase V: consenso do diagnóstico

Finalmente, na sexta fase, mostrada na figura 3.5, é enviado por Multicasting para os outros grupos o diagnóstico do grupo_teste; não se utiliza Broadcast por não haver a necessidade do grupo de diagnóstico receber o resultado. O grupo_diagnóstico envia para todas as outras unidades do sistema o diagnóstico de cada unidade do grupo_teste; salienta-se que, quando os tamanhos do grupo são desproporcionais (grandes e pequenos grupos), temos um número de mensagens geradas muito elevado; com isto deve-se ter o cuidado na proporcionalidade do tamanho dos grupos. Em alguns casos, é necessário a recombinação dos grupos após a reconfiguração do sistema.

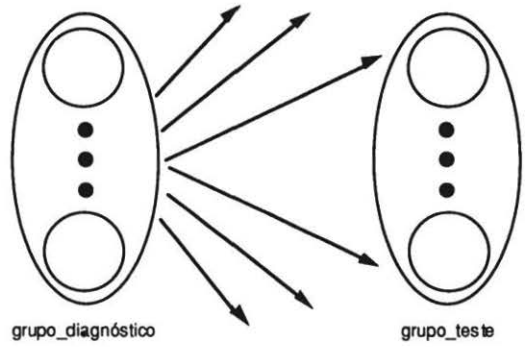

Figura 3.5 - Fase VI: multicasting do diagnóstico

4. Conclusão

Os aspectos de autonomia e independência dos componentes de um SD caracterizam uma série de vantagens em relação a Tolerância a Falhas, tais como: independência de falhas de hardware, maior facilidade de isolamento de erros e maior facilidade de reconfiguração do

sistema após falhas. Para que a reconfiguração seja possível é necessária a identificação dos processadores falhos, este procedimento é denominado de Diagnóstico de Falhas.

Este trabalho apresentou uma proposta de Diagnóstico de Falhas baseada na "Teoria por 1os princípios", que propõem confrontar o comportamento esperado de um sistema com o comportamento observado deste. A partir da detecção de um comportamento anormal, são determinados como candidatos os componentes que podem ter causado este comportamento anormal, a partir deste conjunto são feitos testes e minimizações no conjunto para determinar qual o componente que realmente causou a falha.

O objetivo deste trabalho foi definir um sistema de diagnóstico de falhas que pudesse ser utilizado em ambientes de estações distribuídas, conectadas entre rede. Para isto foi feita uma modelagem do sistema, baseada em "frames". Foi determinado que o grande fator de custo neste processo de diagnóstico é a fase de consenso entre as unidades; com a finalidade de minizar este custo, foi definido grupo de diagnóstico: um grupo de unidades testa outro grupo, e ao final o resultado é fornecido as outras unidades que não pertençam ao grupo_diagnóstico. O protocolo de difusão confiável, determinado para a fase de consenso, também é de grande importância para o processo, visto que alguns geram grande número de mensagens e outros, para diminuir as mensagens, exigem processamento.

Bibliografia

[BAR 90] BARCELLOS,A. et all. **Tolerância a Falhas em Sistemas Distribuídos.** CPGCC/II - UFRGS, Porto Alegre, RS, 1990.

[DUE 92] DUEÑAS, L. **Proposta e Estudo de um Algorítmo de Difusão Confiável para Aplicações Distribuídas Replicadas.** Dissertação para obtenção de Grau de Mestre - UFSC, Florianópolis, SC, 1992.

[DRU 87] DRUMMOND,R. **Sistemas Distribuídos. Introdução a Tolerância a Falhas** - Mini Curso. II Simpósio de Computadores Tolerantes a Falhas, Campinas, 1987.

[REI 87] REITER,R. **A theory of Diagnosis from First Principles.** Artificial Intelligence, v.32, n.1, p.57-95, Amsterdan, april 1987.

[WEB 90] WEBER,T. et all. **Fundamentos de Tolerância a Falhas.** JAI - Jornada de Atualização em Informática, Vitória, Congresso da SBC, 1990, 75 páginas.