CosNamingFT - Um Serviço de Nomes Tolerante a Falhas em Conformidade com o Padrão OMG

Lau Cheuk Lung, Joni da Silva Fraga e Jean-Marie Farines
Laboratório de Controle e Microinformática - LCMI-DAS-UFSC
Campus Universitário - Trindade - Florianópolis - SC
Caixa Postal 476 - CEP 88040-900
e-mail: {lau, fraga, farines}@lcmi.ufsc.br

Resumo

Este artigo apresenta o projeto e a implementação de um serviço de nomes tolerante a falhas - *CosNamingFT*, em conformidade com o padrão OMG (*Object Management Group*). Um objeto CORBA pode ser acessado através da sua referência de objeto IOR (*Interoperable Object Reference*) que é registrado no serviço de nomes. Portanto, o *serviço de nomes* pode ser considerado o "portão de acesso" aos diversos objetos em um sistema distribuído, o que torna desejável que este seja tolerante a falhas. A replicação do *serviço de nomes* é apoiado no pacote *GroupPac* [3], o qual segue a linha de soluções abertas preconizada pela OMG, propondo um conjunto de objetos CORBA de serviço que servem como *building blocks* para a implementação de técnicas de tolerância a falhas. Esse trabalho de certa forma demonstra a utilidade de alguns desses objetos de serviços e ainda, de que é possível se conseguir soluções abertas para o problema de replicação de objetos distribuídos.
***Palavras chave**: serviço de nomes, tolerância a falhas, CORBA e sistemas abertos.*

Abstract

This paper describes the project and implementation of a fault-tolerant naming service - *CosNamingFT*, which follows the OMG (*Object Management Group*) standards. A CORBA object can be accessed through its *Interoperable Object Reference* (IOR) recorded in the naming server. Therefore, in this way, the naming service can be considered as the “gate” of access to all objects in a distributed system, what implies the need of fault-tolerance support in the implementing of a name service. The replication of the name server is supported for *GroupPac* package [3], which follows the approach of open programming stimulated by the OMG. The GroupPac services are composed by a group of object services that works as building blocks for the implementation of fault tolerant applications. This paper, in a certain way, tries to demonstrate the usefulness of some of those object services and to show how important is to get open solutions for the replication problem of distributed objects.
***Keywords**: naming service, fault-tolerant, CORBA e open systems.*

1. Introdução

O serviço de nomes tem um papel muito importante em sistemas distribuídos. O seu principal objetivo é facilitar o acesso a recursos compartilhados. O serviço de nomes, se consideramos o modelo de objetos distribuídos definido através das especificações CORBA (*Common Object Request Broker Architecture*), pode ser tomado como “um portão de acesso” aos diversos objetos em um sistema distribuído. Todos os objetos distribuídos têm acesso ao serviço de nomes pelo menos uma vez durante o seu ciclo de vida: ou para registrar sua referência de objeto ou para obter a referência de outros objetos. No contexto da OMG (*Object Management Group*) [1], foi definido um *COSS* (*Common Object Services Specification*) padronizando o serviço de nomes para o acesso via CORBA a objetos distribuídos. As especificações do serviço de nomes (*CosNaming*) [2] foram padronizadas de forma a considerar aspectos de portabilidade, interoperabilidade e reusabilidade que são conceitos importantes em sistemas abertos. No entanto, essas especificações não contemplam requisitos de tolerância a falhas, fundamentais no caso de sistemas distribuídos.

Uma possível solução para garantir a tolerância a falhas do serviço de nomes seria a utilização do *serviço de persistência* das COSS [2]. Essa especificação define uma interface única para serviços de armazenamento persistente de estados de objetos. O estado do objeto

pode ser tratado considerando duas partes: o estado dinâmico, que tipicamente está na memória principal e é formado por informações que muito provavelmente não estarão disponíveis por muito tempo no ciclo de vida do objeto (esse estado não seria, por exemplo, preservado na ocorrência de uma falha no sistema); e o estado persistente que o objeto usa para reconstruir o seu estado dinâmico. A decisão de não seguir uma abordagem fundamentada no serviço de persistência é devido a fatores de desempenho. Um serviço de nomes com copias persistentes, quando da ocorrência de *crash*, necessita do administrador para reiniciar o mesmo e fazer com que esse recupere seu estado a partir de um arquivo, por exemplo.

A OMG ainda não tem também especificado um serviço ou serviços que suportem alguma forma de processamento replicado ou de tolerância a falhas. Os esforços iniciais foram dados através de um documento *Request for Proposal* (*RFP*), requisitando propostas para a padronização de funcionalidades CORBA para o suporte a aplicações tolerantes a falhas na arquitetura OMA [11]. A idéia básica é fornecer serviços e facilidades para a construção de aplicações tolerantes a falhas. Essas propostas deverão apresentar soluções abertas, seguindo a mesma linha que orienta os padrões CORBA.

Apresentamos neste artigo o projeto e a implementação do *CosNamingFT* - um serviço de nomes que segue as especificações COSS *CosNaming* da OMG e apresenta atributos de tolerância a falhas. O *CosNamingFT* está baseado na técnica de replicação passiva *primário/backups* [4][6] o que garante, mesmo em presença de falhas, o oferecimento de serviço contínuo. As hipóteses de falhas assumidas nesse trabalho se limitam a *crashes*. A replicação do *serviço de nomes* implementada é apoiada no pacote *GroupPac* [3], que está sendo desenvolvido atualmente no nosso laboratório. O *GroupPac* segue essa linha de soluções abertas preconizada pela OMG, propondo um conjunto de objetos CORBA de serviço que servem como blocos de construção ("*building blocks*") para a implementação de técnicas de tolerância a falhas ou de modelos de comunicação de grupo. Esses objetos de serviço podem ser combinados permitindo a implementação de diferentes esquemas de tolerância a falhas, sem depender de soluções proprietárias.

Esse trabalho de certa forma demonstra a utilidade de alguns desses objetos de serviços e ainda, de que é possível se conseguir soluções abertas para o problema de replicação de objetos distribuídos. O artigo apresenta na seção 2 o padrão *CosNaming* da OMG para o serviço de nomes. Na seção 3, o pacote *GroupPac* tem descrito seus objetos de serviço. A arquitetura do *CosNamingFT* é objeto de discussão nas seções 3 e 4. Na seção 5, é apresentada alguns testes de desempenho realizados sobre o *CosNamingFT*. Por fim, na seção 6 são discutidos e comparados com o *CosNamingFT*, os trabalhos relacionados presentes na literatura.

2. Serviço de Nomes no Padrão CORBA

As especificações CORBA/OMG correspondem a um conjunto de padrões e conceitos para objetos distribuídos em ambientes abertos propostos pela OMG (*Object Management Group*)[1]. Segundo essa arquitetura, métodos de objetos remotos podem ser invocados de forma transparente em ambientes distribuídos heterogêneos através de um ORB (*Object Request Broker*). O ORB, num sentido mais genérico, é um canal de comunicação para objetos distribuídos. No CORBA, a interoperabilidade entre objetos desenvolvidos é conseguida a partir das especificações IDL (*Interface Definition Language*) das interfaces de objetos. A tradução de especificações IDL gera as interfaces necessárias em tempo de

[1] É uma organização formada por mais de 700 empresas com o objetivo de especificar um conjunto de padrões e conceitos para a programação orientada a objetos em ambientes distribuídos abertos.

execução para a ativação de métodos, independente das linguagens de programação (C, C++, Ada, Cobol, Java, etc.) e suportes usados nas implementações dos objetos. A IDL é uma linguagem declarativa sem nenhuma estrutura algorítmica, com sintaxes e tipos predefinidos baseados na linguagem C++.

No desenvolvimento de aplicações orientadas a objetos o padrão CORBA/OMG oferece ainda um conjunto de objetos de serviço que facilitam o trabalho do projetista da aplicação. Esses serviços são padronizados pela OMG dentro da perspectiva da arquitetura OMA (*Object Management Architecture*). Os objetos de serviço COSS (*Common Object Services Specifications*) formam uma coleção de serviços (interfaces e objetos) a nível de sistema que oferecem funções básicas para o uso e a implementação de objetos de aplicação. As especificações COSS podem ser entendidas como extensões ou complementações das funcionalidades do ORB.

2.1 *CosNaming*

Dentre as especificações COSS aparece o *serviço de nomes* (*CosNaming*) que define os meios para a resolução de nomes no ambiente CORBA; o que inclui a administração distribuída e heterogênea de nomes e seus contextos. Essas especificações definem ainda orientações no sentido das implementações necessárias desse serviço.

Um objeto é localizado fazendo uso de seu *nome* e da sua *referência de objeto*. O nome de um objeto corresponde a um conjunto de caracteres normalmente expressando uma qualidade ou característica associada ao mesmo. Na referência de objeto estão contidas as informações necessárias para a localização do objeto em um sistema distribuído. Nas especificações *CosNaming*, uma associação entre nome e referência de objeto é chamada de *name binding*. Um *name binding* é sempre definido dentro de um contexto de nomes (*name context*). Cada objeto CORBA *NamingContext* manipula um conjunto próprio de *name bindings* em que cada nome é único, mas que não impede, caso desejado, a associação de diferentes nomes a uma única referência de objeto. O termo *NamingContext* será usado no decorrer desse texto como sendo o objeto que implementa as funcionalidades do serviço de nomes definidas nas especificações *CosNaming*.

Na OMG, a referência de objeto é definida numa forma padrão conhecida como IOR (*Interoperable Object Reference*). A IOR [2] é uma sequência de caracteres que quando convertido para o formato adequado fornece as informações de endereço IP do sítio, a porta, um ponteiro para o objeto a ser acessado e algumas informações de controle relacionadas da própria IOR. Cada objeto distribuído deve ter sua IOR disponível, registrada em um *NamingContext*. Com isso, um cliente necessitando da IOR de um determinado objeto servidor deve, através do envio do nome do mesmo, obtê-lo no *NamingContext* desse objeto.

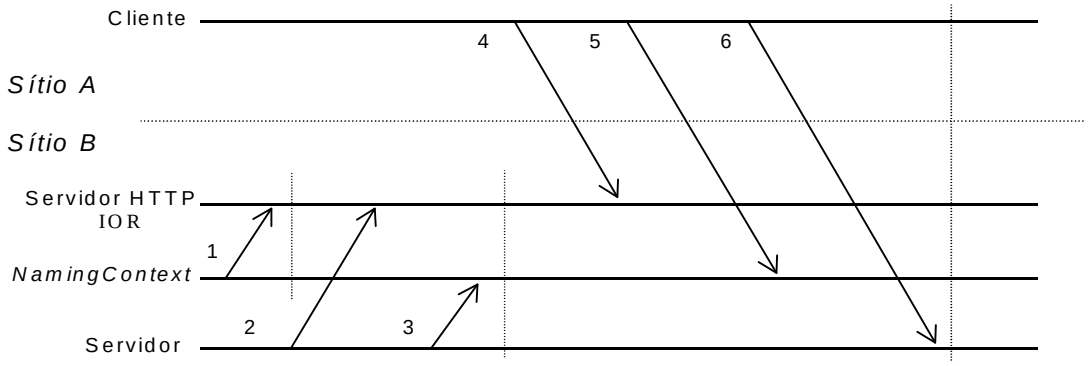

Figura 1. Os procedimentos do *NamingContext* no suporte às interações cliente/servidor.

A figura 1 apresenta um diagrama temporal separando em fases os procedimentos no suporte dado pelo serviço de nomes (*NamingContext*) às interações cliente/servidor. Essas fases compreendem a iniciação do *NamingContext*, registro de uma IOR de objeto servidor no *NamingContext* (*binding*), e por último, a obtenção, de uma IOR de objeto servidor por parte de um cliente.

Um objeto *NamingContext* na sua iniciação (passo 1, figura 1) deve colocar sua própria IOR num repositório (um arquivo em um sistema de arquivos), tornando-a disponível a objetos CORBA em geral para que possam acessá-lo. O serviço de nomes descrito nesse texto, tem a sua IOR colocada num diretório com acesso através de um servidor HTTP (ex: http://www.lcmi.ufsc.br/~lau/ior) para facilitar invocações sobre esse serviço via *internet*. Uma vez o *NamingContext* em execução os objetos CORBA podem registrar suas IORs e se tornarem disponíveis no sistema.

Todo objeto servidor quando de sua iniciação registra sua IOR em um *NamingContext*. Para isso, o servidor precisa primeiro obter a IOR do *NamingContext* o que é feito através do seu método `resolve_initial_reference`, que tem a função de acessar o repositório (servidor de HTTP no nosso caso) para obter a IOR do *NamingContext* (passo 2 da figura 1). O código em Java que implementa este passo é apresentado na figura 2.

```
public class FooServer {
  public static void main(String args[]) {
    try {
      // obtendo o IOR do servidor de nomes
      NamingContext ncRef =
         (NamingContext)ORB.resolve_initial_references("NameService");
               :
```

Figura 2. Obtendo o IOR do *NamingContext*.

Após obter a IOR do *NamingContext*, o servidor da aplicação pode invocá-lo para se registrar e se tornar disponível no sistema (passo 3 da figura 1). O código da figura 3 mostra a definição do nome do objeto a ser registrado na estrutura *NameComponent* que consiste de dois atributos: um identificador (ex: "Foo") e um tipo (ex: "App"). Uma vez definido o *NameComponent* o servidor chama o *NamingContext* para registrar o par nome e referência de objeto usando o método `bind`.

```
          :
      // ref – referência do objeto servidor Foo
      Foo ref = new FooServant();
      NameComponent path[] = {new NameComponent("Foo", "App")};
      // bind a referência de objeto no NamingContext
      ncRef.bind(path, ref);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
```

Figura 3. Registrando no *NamingContext*.

Os procedimentos de um cliente no sentido de viabilizar a sua comunicação com um objeto servidor são mostrados na figura 1, nas interações indicadas como 4, 5 e 6. Uma vez que um objeto servidor está disponível no sistema, o cliente para ter suas requisições de métodos executadas sobre esse servidor deve primeiro obter a IOR do *NamingContext* usando o seu método `resolve_initial_reference` (passo 4 da figura 1). Com a posse dessa IOR (disponível a partir do servidor de HTTP), o cliente usando o nome do objeto servidor a ser acessado monta o `NameComponent("Foo", "App")` e invoca o *NamingContext* para resolver esse nome (passo 5 da figura 1). Se o nome existir dentro do contexto indicado, o

serviço retorna a IOR associada a esse nome enviado pelo cliente. A partir daí o cliente pode desempenhar as invocações no servidor (passo 6 da figura 1). A figura 4 apresenta o código que implementa as interações do cliente para viabilizar a sua comunicação com o objeto servidor.

```
public class Client {
  public static void main(String args[]) {
    try{
      NamingContext ncRef =
         (NamingContext)ORB.resolve_initial_references("NameService");
      // obtém a referência do objeto Foo (ref) a partir do seu nome.
      NameComponent path[] = {new NameComponent("Foo", "App")};
      Foo ref = FooHelper.narrow(ncRef.resolve(path));
      // invoca o método m0 do objeto Foo
      ref.m0();
      System.exit(0);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
```

Figura 4. Interações do cliente para viabilizar invocações sobre um objeto servidor.

3. Pacote de Serviços *GroupPac* e a Arquitetura do *CosNamingFT*

3.1 O Pacote de Serviços *GroupPac*

A OMG ainda não tem especificado um serviço de suporte a processamento replicado ou de tolerância a falhas. Existem somente esforços de um primeiro documento *Request for Proposal* (RFP) requisitando propostas para a padronização de funcionalidades CORBA de suporte a aplicações tolerantes a falhas na arquitetura OMA [11]. A idéia básica é fornecer serviços e facilidades para a construção de aplicações tolerantes a falhas. Essas propostas deverão apresentar soluções abertas, não dependentes de protocolos ou ferramentas proprietárias como as apresentadas em [14]. O *GroupPac* [3] que está sendo desenvolvido atualmente no nosso laboratório, segue essa linha de soluções abertas, propondo um conjunto de objetos CORBA de serviço que fornecem suporte à aplicações na construção e gestão de replicações. Esse pacote de serviços para objetos distribuídos é baseado nos modelos e conceitos definidos pela OMG para objetos de serviço [2]. Conceitualmente, dentro dessa filosofia de objetos de serviço, o *GroupPac* fornece um conjunto de blocos de construção ("*building blocks*") - os objetos de serviço - que podem ser arranjados de diferentes formas no sentido de compor diferentes esquemas ou arquiteturas de serviços de aplicação com propriedades de tolerância a falhas. Todos os objetos de serviço são ortogonais e especificados em suas interfaces através da IDL do padrão CORBA.

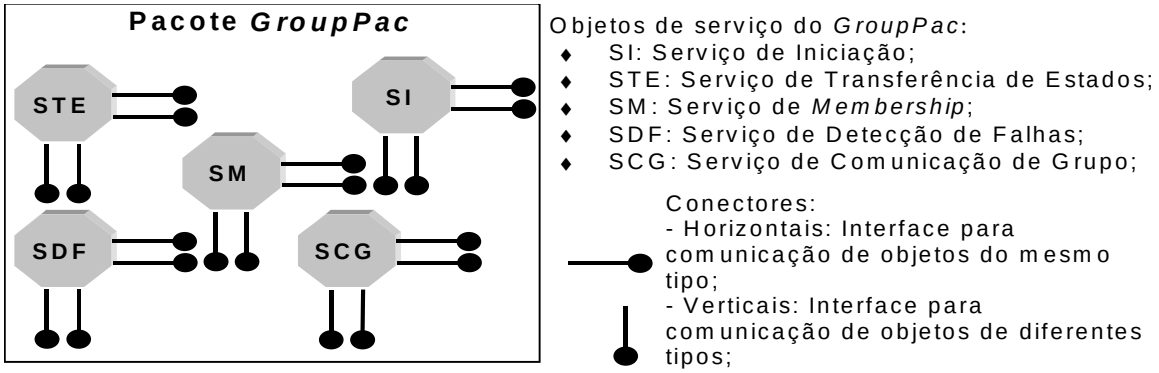

Figura 5. O pacote de objetos de serviço do *GroupPac*.

Na figura 5, são apresentados os objetos de serviço SI, STE, SM, SDF e SCG - todos fazendo parte do *GroupPac*. Cada um desses objetos possui uma interface horizontal e uma vertical: a primeira permite as comunicações entre objetos de serviço do mesmo tipo, e a segunda constitui-se no meio pelo qual fornecem serviços para outros objetos.

O objeto SI (serviço de iniciação) é responsável pela criação e iniciação dos outros objetos de serviço do pacote. É a partir desse objeto que a configuração necessária ao suporte de uma aplicação replicada é construída. O objeto STE (serviço de transferência de estado) oferece funcionalidades para transferências de estados de um objeto para outro. Esse serviço é usado, por exemplo, em modelos de replicação ou ainda para migração de objetos. O objeto SM (serviço de *membership*) é responsável pelo serviços de gestão de grupos de réplicas (grupos de objetos). Essa gestão deve ser transparente à aplicação, exercendo um controle dinâmico nas entradas e saídas de objetos de um grupo pela manutenção de listas atualizadas de seus membros (*membership*). O objeto SDF (serviço de detecção de falhas) envolve um conjunto de procedimentos de detecção de falhas de objetos em um grupo. O SDF opera em conjunto com o objeto SM: quando o SDF detecta um *crash* de um dos objetos do grupo, essa falha é imediatamente reportada ao objeto SM para que esse gere uma nova lista de membros. Por último, temos o objeto SCG (serviço de comunicação de grupo) que oferece um conjunto de facilidades para comunicação de grupo. Este serviço fornece um conjunto de protocolos confiáveis de comunicação de grupo, com diferentes políticas de ordenação de mensagem (FIFO, Causal ou Total), construídos a partir de alguns objetos de serviço (por exemplo, o SM) e de comunicações simples, ponto a ponto, a nível de ORB.

Esses serviços descritos acima devem ser combinados permitindo a implementação de diferentes esquemas de tolerância a falhas, sem depender de soluções proprietárias. Em [16] é desenvolvida uma abordagem similar a implementada no *GroupPac*, mas que se limita somente a definir objetos de serviço que combinados fornecem suporte de comunicação confiável para grupos de objetos no ambiente CORBA.

3.2 Arquitetura do *CosNamingFT*

A implementação do *CosNamingFT* segue as especificações do padrão COSS de um serviço de nomes definido pela OMG [2] adicionando a esses atributos características de tolerância a falhas. Para a replicação do *NamingContext* foi usada a abordagem de objetos de serviço do *GroupPac*. Esses objetos de serviço são usados para implementar as funcionalidades do modelo *primário/backups* [10]. A implementação do *CosNaming* replicado usando esse modelo de replicação passiva, define um objeto *primário* (um *CosNaming* ativo) e as réplicas restantes como *backups* (objetos *CosNaming* passivos). As comunicações dos clientes dos serviços do *CosNaming* são todas realizadas com a réplica primária. A replicação usada é não bloqueante [10]: o primário tem a função, sem se bloquear, de processar a requisição de serviço, enviar mensagens de atualização de estado (*checkpoints*) aos objetos *backups*, e por fim responder ao cliente emissor da requisição com os resultados do processamento. Os *backups* podem assumir o papel de primário em caso de falha desse último. Após o processamento de cada requisição de cliente e as respectivas atualizações de estado, todas as réplicas corretas que formam o *CosNamingFT* devem se manter com os mesmos *name bindings*. A seguir damos uma descrição de como e porque cada objeto de serviço é incorporado na arquitetura *CosNamingFT*.

Os objetos de serviço do *GroupPac* são incorporados no *CosNamingFT* de forma transparente (figura 6). Cada réplica do *CosNamingFT* é composta pelos objetos *NamingContext*, SI, STE, SM e SDF e, em termos de modelo de execução, são conformados em um único processo UNIX. A falha de um destes objetos é considerado como falha da réplica do *CosNamingFT* como um todo (*crash* da réplica). Os serviços STE, SM e SDF, são

compostos na forma de grupos de objetos de serviço. Por exemplo, levando em conta as conexões horizontais da figura 6, os objetos SM de cada réplica *CosNamingFT* formam o grupo que fornece serviço de *membership*.

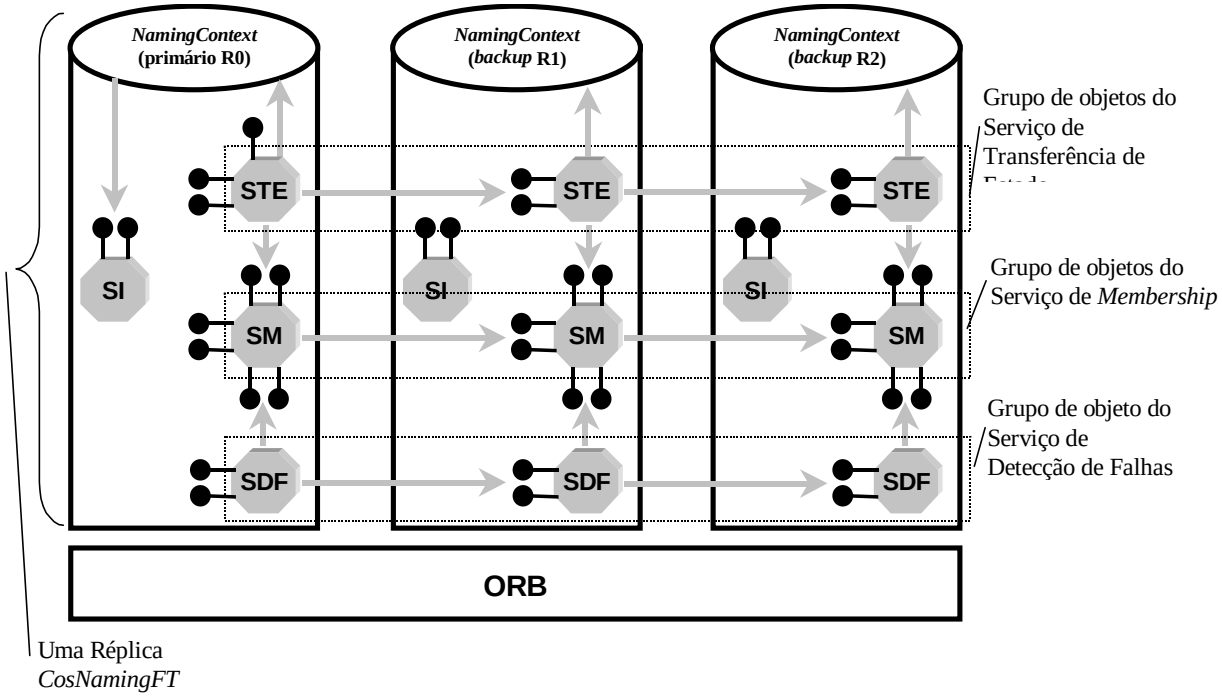

Figura 6. A composição dos blocos de serviço para formar a arquitetura do *CosNamingFT*.

3.2.1 Serviço de Gestão

Para dar suporte ao modelo de replicação *primário/backups* é necessário um serviço de gestão que forneça informação de quais réplicas do *CosNamingFT* estão operacionais e que também controle as entradas e saídas dinâmicas das réplicas. A combinação dos grupos SM e SDF suprem essas necessidades .

O serviço de detecção de falhas implementado pelo grupo SDF tem a finalidade de detectar falhas de *crash* de réplicas *CosNamingFT*. Esse serviço implementa um protocolo com premissas de comunicações ponto-a-ponto, confiáveis e assíncronas entre os pares comunicantes. Na figura 7, cada objeto do grupo SDF envia mensagens do tipo "você está ativo ?", periodicamente (período T(s)), ao membro anterior da ordenação em anel virtual do grupo. Ao não receber resposta de seu antecessor dentro de um prazo pré-definido, uma réplica R_i considera a possibilidade do *crash* de seu antecessor R_{i-1}, devendo então ativar o método `leave_group` do objeto SM da réplica primária. Essa ativação indica a suspeita em relação a réplica R_{i-1}.

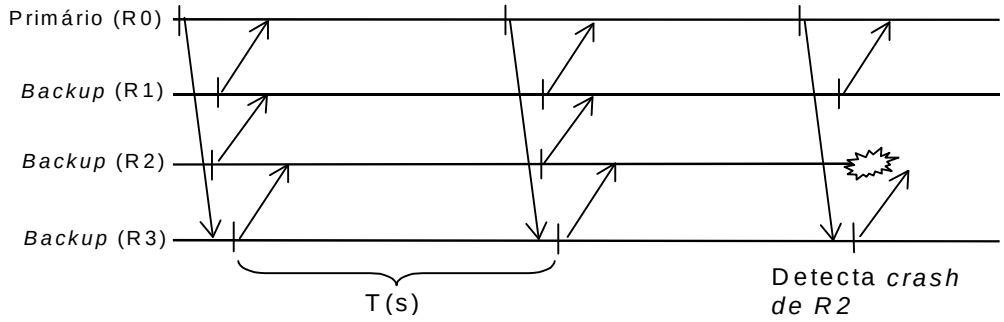

Figura 7. O serviço de detecção de falhas.

O serviço de *membership* é então executado (ou seja, os objetos do grupo SM são ativados). O protocolo implementado nesse serviço é de decisão centralizada envolvendo interações entre as réplicas em duas fases [13]. A coordenação na obtenção de uma nova lista de membros (*new view*) é centralizada no objeto SM da réplica primário do *CosNamingFT*. A figura 8 mostra um exemplo de funcionamento desse protocolo. O objeto SDF da réplica R3 ao detectar o *crash* da réplica R2 envia a mensagem "suspeite de R2" para o objeto SM da réplica primário. Neste ponto, o protocolo de *membership* é ativado e o primário difunde uma mensagem *commit* para detectar quem ainda continua no grupo, as réplicas que estão ativas devem dar um reconhecimento a esta mensagem (*Ack*). Após um prazo de espera pré definido, o primário produz uma nova lista de membros a partir dos *Ack* recebidos [4][13].

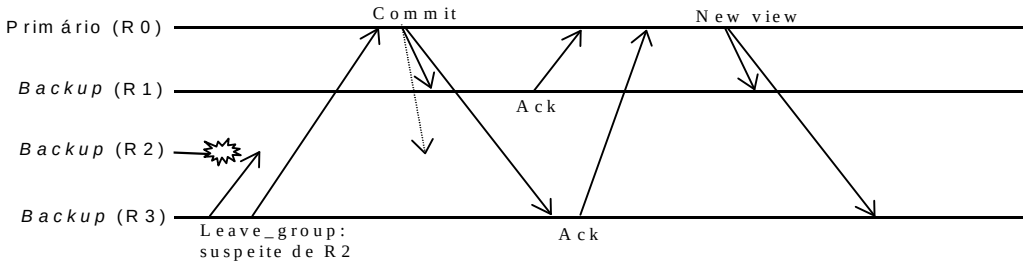

Figura 8. Situação: *crash* da réplica R2.

Quando ocorre o *crash* da réplica primário do *CosNamingFT*, o novo primário é escolhido entre as réplicas *backups*. A definição do novo primário é feita no anel virtual, recaindo sempre sobre o membro mais antigo do grupo (o sucessor do antigo primário). Nesse modelo de replicação passiva, apenas a réplica Primário é ativa e interage com os objetos clientes. Portanto, somente a IOR da réplica Primário do serviço de nomes precisa estar disponível, a partir de um repositório do sistema (no nosso caso o servidor HTTP, item 2), a todos os objetos de aplicação. Na substituição do primário, o novo primário deve na sua iniciação registrar o seu IOR no servidor HTTP. A figura 9 exemplifica a detecção da falha de uma réplica primária e a sua substituição por uma das réplicas *Backup*. Nesse exemplo R3 suspeita de R0 e ativa o método `leave_group` do objeto SM na réplica R1. Essa última réplica passa então a centralizar a execução do protocolo de *membership*.

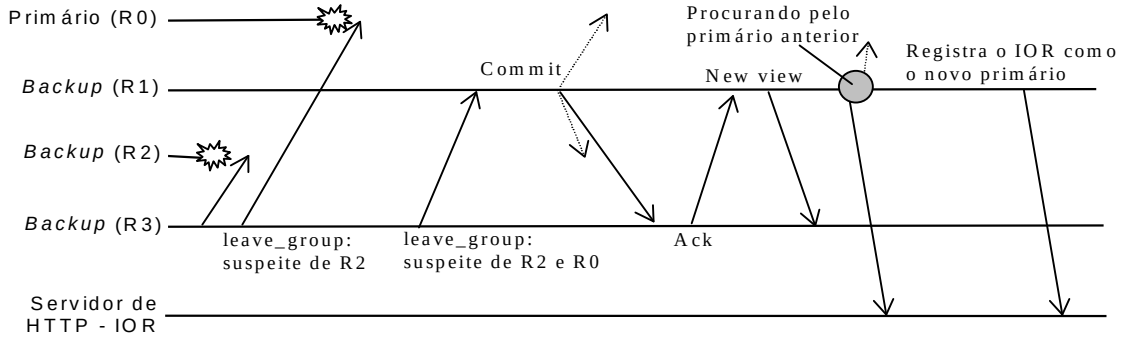

Figura 9. Situação: *crash* da réplica R2 e R0 e a escolha do novo primário.

A réplica primário é sempre aquela que tem a sua IOR registrada no repositório (servidor HTTP). Quando um novo primário é eleito, esse na sua iniciação lê o IOR disponível no servidor HTTP antes de registrar o seu (apagando o antigo). De posse desse IOR lido, a réplica tenta se comunicar com o primário anterior (aquele que supostamente está em *crash*). Caso receba resposta, a réplica que está tentando se registrar como novo primário, deve cessar todo o seu processamento de iniciação e se reintegrar ao grupo como uma réplica *Backup*. No caso de não haver resposta, o réplica se registra no servidor de HTTP se tornando

efetivamente o primário do grupo (R1 da figura 9). O procedimento descrito evita a possibilidade de se ter duas réplicas primários em um dado instante.

3.2.2 Serviço de Transferência de Estado

O serviço de transferência de estado (STE) do *GroupPac* é também incorporado às réplicas que formam o *CosNamingFT*, para garantir que cada *backup* mantenha seu estado consistente em relação ao primário. As transferências de estados do primário não são bloqueantes. Na figura 10, são apresentados os passos executados por esse serviço para a transferência do estado primário para os *backups* na invocação do método `bind` (operação de escrita).

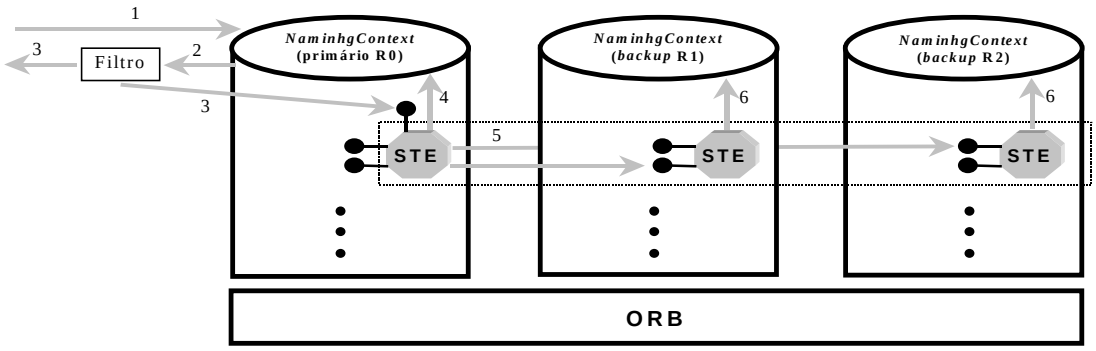

Figura 10. O serviço de transferência de estados.

Passos:

1. Invocação de um cliente sobre o serviço *NamingContextFT*, que deve ser executada sobre a réplica primário desse serviço;
2. A resposta correspondente à invocação é interceptada;
3. A referência do objeto interceptado é enviada ao serviço de transferência de estados para que este possa invocar o objeto *NamingContext* primário.
4. A resposta interceptada retoma ao seu caminho natural de envio ao cliente;
5. Com a referência do objeto *NamingContext* primário, o STE invoca esse objeto obtendo o seu estado (get_state);
6. Com as informações de estado, o objeto STE local difunde o estado do primário para os seus pares de grupo nas réplicas *backups*;
7. Cada objeto STE local ao receber o estado do primário executa o *upcall* set_state no sentido de atualizar o objeto *NamingContext* de sua réplica;

4. Implementação do *CosNamingFT*

Discutiremos nesta seção alguns aspectos relativos a implementação de cada um desses blocos de construção que compõe a arquitetura do *CosNamingFT*. A implementação do *CosNamingFT* adota as especificações COSS/OMG referente a serviço de nomes (*CosNaming*). Nenhuma extensão ou modificação é feita na interface padrão, como mostrado na figura 11. A inclusão dos serviços do *GroupPac* também em nada altera as especificações COSS/OMG. Todos os serviços deste pacote são implementados em *Java* e suas interfaces especificadas segundo o padrão IDL/OMG. Todas as implementações realizadas tiveram como plataforma de desenvolvimento o *OrbixWeb* [5], produto comercial desenvolvido pela IONA *Technologies*, Ltd. Esse *middleware* corresponde a um conjunto de suportes e ferramentas de desenvolvimento que permitem construir e integrar aplicações orientadas a objetos distribuídas em ambientes heterogêneos. O *OrbixWeb* é completamente desenvolvido

na linguagem *Java*, trazendo as vantagens desta linguagem, seguindo as especificações da OMG. Todos os objetos devem ter suas interfaces bem definidas em uma IDL que é compatível com as especificações CORBA/OMG. Todos os componentes do *OrbixWeb* e aplicações se comunicam usando o protocolo IIOP, também padronizada pela OMG [2].

4.1 Estrutura Geral

A implementação do *CosNamingFT* envolve dois componentes de *software* codificados em Java: o *NamingContextServant.java* e o *NamingContextServer.java*. O componente *NamingContextServant.java*, por sua vez, implementa as operações do *CosNaming* descritas nas especificações IDL. A interface do *NamingContext*, implementada nesse componente de *software*, é apresentada na figura 11, sendo basicamente constituída pelas operações de conectar (*bind*), reconectar (*rebind*) ou desconectar (*unbind*) um nome a um objeto, operações de recuperar um objeto através do seu nome (*resolve*), e ainda de operações para criar, destruir ou listar um contexto de nomes. A essas operações implementadas no *NamingContextServant.java*, são adicionadas ainda operações de *upcall* como o `get_state` e o `set_state`, necessárias para o fluxo de informações entre os objetos de serviço CORBA do *Grupac* e o serviço de nomes.

O componente *NamingContextServer.java* implementa as funcionalidades que envolvem a criação e iniciação dos objetos que compõem as réplicas no *CosNamingFT*. A implementação do *NamingContextServer.java* segue de uma forma geral as especificações, adicionando ainda códigos para o registro da IOR do objeto primário no servidor HTTP e, também, para a iniciação do objeto SI (serviço de iniciação) para que este crie e inicie os objetos de serviço do *GroupPac* na réplica correspondente.

```
module CosNamingFT {
   typedef string Istring;
   struct NameComponent {
      Istring id;
      Istring kind;
   };
   typedef sequence <NameComponent> Name;
   enum BindingType { nobject, ncontext };
         :
   interface NamingContext {
         :
      void bind(in Name n, in Object obj)
       raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
    void rebind(in Name n, in Object obj)
         raises(NotFound, CannotProceed, InvalidName);
      void bind_context(in Name n, in NamingContext nc)
         raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
      void rebind_context(in Name n, in NamingContext nc)
         raises(NotFound, CannotProceed, InvalidName);
      Object resolve(in Name n)
         raises(NotFound, CannotProceed, InvalidName);
      void unbind(in Name n)
         raises(NotFound, CannotProceed, InvalidName);
      NamingContext new_context();
      NamingContext bind_new_context(in Name n)
         raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
      void destroy()
         raises(NotEmpty);
         :
};
```

Figura 11. A IDL do *CosNaming*

4.2 A IDL dos Serviços do *GroupPac*

A IDL dos serviços do *GroupPac* é apresentado na figura 12. Cada uma dessas interfaces será representada por um objeto de serviço em tempo de execução. Uma réplica *CosNamingFT* é constituída por um objeto *NamingContext* e pelos objetos do *GroupPac* responsáveis pela gestão do modelo de replicação implementado. Cada réplica do *CosNamingFT* é mapeada em um processo UNIX na nossa implementação.

Como cada serviço do *GroupPac* é um conjunto de objetos CORBA, as comunicações remotas entre os mesmos são também através do ORB. Na nossa implementação, no sentido de facilitar, as referências de objeto que envolvem esses serviços têm como repositório o serviço de *membership* (cada objeto SM que compõem este grupo mantém uma cópia do conjunto dessas referências na estrutura `AllServRefs`). O acesso a essas referências de objeto é local e garantido através de herança da interface *MembershipService*. Com isso, cada objeto de serviço desejando realizar uma interação com seus pares complementares de grupo podem fazer através da obtenção da IOR necessária junto ao objeto SM localizado na sua réplica do *CosNamingFT*.

A estrutura `ServiceRefs` é utilizada pelo objeto SI no momento em que um processo (uma réplica) *CosNamingFT* solicita a entrada no grupo replicado. Após iniciar os objetos de serviço, o objeto SI invoca o método `join_group` do objeto SM enviando a estrutura `ServiceRefs`, que contém as referências de cada objeto de serviço (fd, st e ms da figura 12), pertencente ao processo solicitante. Após o protocolo de *membership* ter sido executado e decidido que o novo processo pode ser inserido no grupo *CosNamingFT*, o objeto SM primário difunde o novo `AllServRefs` para todos os objetos SM do grupo invocando o método `view_change`. O método `leave_group` é invocado no SM do primário pelos objetos do grupo de detecção de falhas (SDF) na detecção de um *crash* de uma réplica do grupo (seção 3).

```
module GroupPac {
  typedef sequence<any> State;
  typedef sequence<string> MembersCrash_seq;
    :
  // Membership service IDL interface - SM
  interface MembershipService {
    struct ServiceRefs {
      Object fd;
      Object st;
      MembershipService ms;
    };
    :
    struct AllServRefs {
      sequence<string> id_seq;   // id: Identifier
      sequence<Object> fd_seq;   // fd: Failure Detector Reference
      sequence<Object> st_seq;   // st: State Transfer Reference
      sequence<MembershipService> ms_seq; // ms: Membership Reference
    };
    :
    boolean join_group(in ServiceRefs servRefs, out string rank);
    boolean leave_group(in MembersCrash_seq membersCrashId);
    void view_change(in short newRank, in short new_view_number);
    void commit(in short leaderRank, in string leaderId,
                in MembershipService leaderRef);
    boolean ack(in string memberAck);
  };
```

```
  // Failure Detector IDL interface - SDF
  interface FailureDetector: MembershipService {
    void keep_alive();
  };

  // State Transfer IDL interface - STE
  interface StateTransfer: MembershipService {
    void get_state(in Object obj, in string stg);
    void set_state(in short id, in State state);
  };
};
```

Figura 12. A IDL dos serviços utilizados do *GroupPac*

A implementação dos objetos detetor de falhas (SDF) se fundamenta basicamente no método `keep_alive`. Cada membro do grupo SDF invoca esse método no membro anterior da sequência de anel virtual do grupo. A detecção de *crash* ocorre quando uma exceção do CORBA indicando falha de comunicação é sinalizada.

4.3 Detalhes de implementação do STE

Em relação as transferências de estado para os *backups*, executadas através dos objetos STEs, é preciso que se defina quais as informações da réplica primário são necessárias para manter a consistência entre réplicas no modelo *primário/bakups*. Em cada objeto *NamingContext*, os pares *name binding*, formados pela associação de cada nome (*NameComponent*) à referência de objeto (IOR) correspondente, são guardados na variável `context`. Essa variável `context`, implementada usando o *hashtable* do pacote java.util.hashtable, constitui o estado da réplica primário definido para as transferências de estados nos casos em que novos *backups NamingContext* são inseridos dinamicamente. Transferências do primário após processamentos referentes a operações que resultam em atualizações do contexto de nomes (como *bind*, *rebind* e *unbind*, por exemplo), resultam em informações de estado que se resumem a apenas um *name binding*.

Os métodos `get_state` e `set_state` (implementados no *NamingContextServantFT*) são métodos que tem a função de converter/desconverter o estado de um objeto para/de um formato de dados padrão definido para as transferência de estado através dos objetos STEs. Devido ao fato de `context` ser um tipo *hashtable*, não definido no CORBA, a solução adotada foi convertê-lo em uma sequência de *bytes*. Essa conversão do `context` para um sequência de *bytes* possibilita a manipulação do mesmo como um tipo `Any` (tipo suportado pelo CORBA). Ou seja, quando o objeto STE invoca em *upcall* o método `get_state` do objeto *NamingContext* (passo 5 da figura 10), esse método pega o `context` e separa cada *name binding* contido nele, convertendo-os em sequências de *bytes*. Na figura 13 é dado um exemplo onde um `context` com dois *name bindings*, em formato de sequência de *bytes*. Essa sequência de *bytes* é definida da seguinte maneira: o primeiro *byte* descreve o número total de *name bindings* (*2*), o *byte* seguinte define o número de *bytes* do primeiro *NameComponent* (*120*), em seguida os *bytes* do *NameComponent* (NC1). Depois, um outro *byte* define o número de *bytes* da primeira referência de objeto (*150*), em seguida os *bytes* da referência de objeto (RefObj1) e assim sucessivamente para os *name binding* subsequentes.

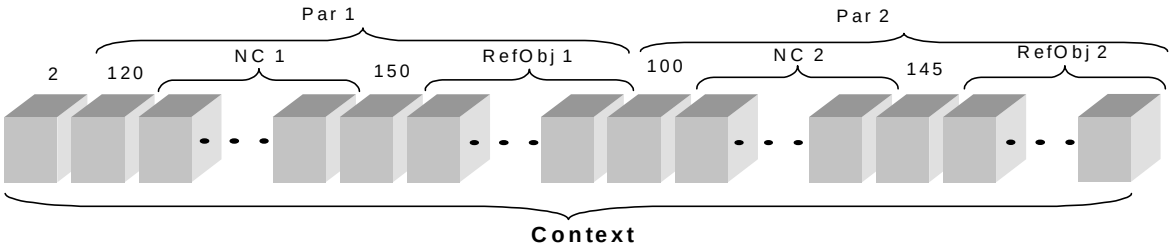

Figura 13. Exemplo de um *context* em uma sequência de *bytes*.

Para a implementação da interceptação descrita na figura 10 (passo 2) é usado o mecanismo de filtro fornecido pelo *OrbixWeb*. O filtro é um dispositivo que provoca o desvio transparente do fluxo normal de uma invocação de método. Esse desvio pode ser usado para adicionar funcionalidades ou controles a uma chamada de método. O serviço de transferência de estados (grupo de STEs) do *CosNamingFT* é ativado usando o filtro em uma interceptação *post-marshaling* para pegar a resposta do objeto servidor e extrair a referência do objeto *CosNaming* da réplica primário, isso tudo de maneira transparente.

5. Análise de Desempenho do *CosNamingFT*

Neste item apresentamos as medidas de desempenho do *CosNamingFT*, realizadas em nosso laboratório, com o sentido de validar nossas propostas. Essas medidas foram realizadas a partir das invocações dos métodos *bind* e *resolve*, portanto, registrando e obtendo a referência de objeto, respectivamente. O desempenho do *CosNamingFT* foi levantado considerando diferentes graus de replicação. O ambiente considerado nesses testes de desempenho foi constituído de uma rede local (com 10 Mbps *Ethernet*) heterogênea usando duas máquinas Sun Ultra 1 com Solaris 2.5, uma Axil 240 também com Solaris 2.5, um Pentium 100 e um Pentium 233 MMX, ambos usando Linux e por fim, um Pentium 233 MMX com Windows 95.

Os testes realizados consistiram de cem (100) chamadas da operação *bind* e também o mesmo número de chamadas da operação *resolve*, todas executadas sobre o *CosNamingFT*. Na figura 14, os tempos de resposta (eixo Y) pelo grau de replicação (eixo X) foram determinados considerando a média dessas cem invocações, nos dois tipos de chamadas. A curva referente a operação *bind* representa um aumento no tempo de resposta a medida que se cresce o grau de replicação do *CosNamingFT*. Esse resultado é explicado pelo fato de que essa operação altera o estado do objeto *NamingContext* primário e que torna necessário, portanto, o envio da atualização de estado às réplicas *backups*. A operação *resolve* apresenta um desempenho praticamente constante em função do grau de replicação. Por ser uma operação de apenas leitura, a operação *resolve* não envolve necessidade de envio de atualizações de estado aos *backups* e portanto tem o desempenho inalterado mesmo quando o número de réplicas cresce.

Nos experimentos realizados com a operação *bind*, o fator de crescimento do tempo de resposta se manteve praticamente constante e de valor aproximado a 8 milisegundos por réplica adicionada ao *CosNamingFT*. Esse fator de crescimento na curva do *bind* é válido para até cinco réplicas. Para um grau de replicação envolvendo seis réplicas, o tempo de resposta medido foi de 54.6 milisegundos. No caso, a taxa de crescimento maior referente a essa sexta réplica foi creditada ao fato que essa sexta réplica foi implementada em uma plataforma bem mais lenta que as demais (um Pentium 100). Se considerarmos um serviço de nomes usual, um grau de replicação considerado razoável para esse serviço seria de três réplicas [14]. Necessidades de graus maiores ou ainda reposições nesse serviço poderiam usar o suporte que permite uma variação dinâmica no número de réplicas – novas réplicas podem ser inseridas

dentro do grupo durante a execução do serviço. Considerando a natureza dessa aplicação, o tempo de resposta de 21.2 milisegundos obtido neste experimento (*bind*) para um grau de replicação três pode ser tomado como bastante aceitável.

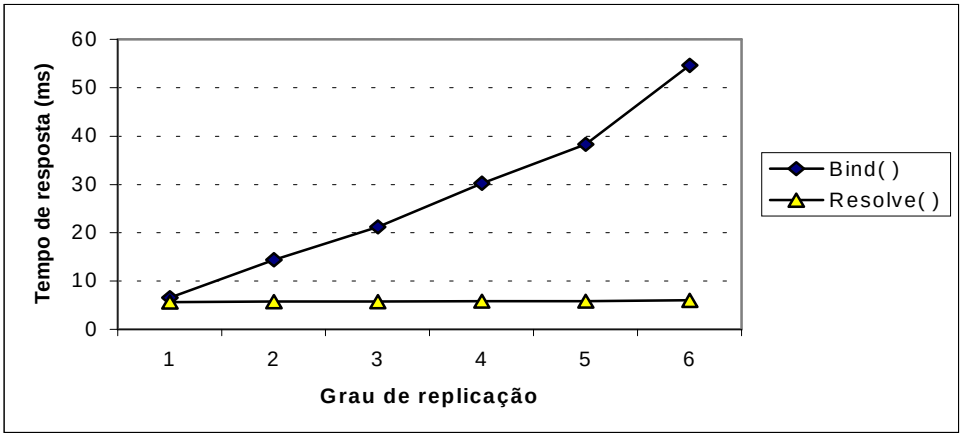

Figura 14. Desempenho do *CosNamingFT*.

6. Trabalhos relacionados

Em [14] é apresentada uma proposta de um serviço de nomes tolerante a falhas usando conceitos CORBA. A técnica de replicação usada nesse trabalho, é baseada na abordagem de replicação ativa: todas as réplicas processam as requisições e a réplica mais rápida retorna a resposta ao cliente. A solução apresentada nesse artigo não é aberta mas, na verdade, é dependente de um suporte de comunicação de grupo de baixo nível tal como Isis [15] ou Horus [4], constituindo-se, dessa maneira, em uma solução baseada em suportes proprietários.

Em [14] são também apresentadas medidas de desempenho para a abordagem de replicação usada no serviço de nomes. Nos experimentos citados, a operação *bind* apresenta uma taxa de crescimento do tempo de resposta por réplica adicionada em torno 6 milisegundos, quando o suporte de baixo nível usado é o Isis. Se o Horus for utilizado como serviço de baixo nível essa taxa fica em torno de 1 milisegundo por réplica por réplica adicionada. O conjunto de máquinas usadas nessas medidas foram cinco *SPARCstation* 10 e quatro *SPARCstation* 20, tendo todas o sistema operacional SUNOS 4.1.3 e considerando as mesmas sobre uma rede de 10 Mbps *Ethernet*. Apesar das dificuldades para efetuar as comparações devido as diferentes opções de máquinas, sistema operacional, técnica de replicação etc., adotados em [14], podemos afirmar que o uso da abordagem de replicação ativa baseado num suporte de comunicação de grupo otimizado é certamente uma justificativa para o melhor desempenho em relação ao nosso trabalho. Mas, por outro lado, podemos citar como uma das vantagens do *CosNamingFT* a sua conformidade com as recomendações de objetos de serviço COSS da OMA. As soluções adotadas no *CosNamingFT* estão baseadas no conceito de sistemas abertos, bem dentro da linha perseguida na OMG para serviços tolerantes a falhas, ou seja, as soluções para tolerância a falhas não devem depender de qualquer suporte ou protocolo proprietário de mais baixo nível.

No nosso laboratório, foi desenvolvida outra experiência de replicação de objetos disponíveis via CORBA [7] [8] [9]. Essa experiência era fundamentada em conceitos da reflexão computacional. O paradigma reflexivo usado seguia o conceito de meta-objetos [17]. A gestão do modelo de replicação definidas nos protótipos não era implementada usando facilidades ou serviços de objeto CORBA mas sim, faziam parte da aplicação na forma de meta-objetos. Esse modelo de implementação de replicações se mostrou bastante flexível, mudar de técnica de replicação se resumia a simples troca de meta-objetos. As soluções

adotadas nesse modelo também estavam fundamentadas em suportes de grupo proprietários. O Isis [15] com a sua disponibilidade de serviços de grupo era parte fundamental do modelo desenvolvido.

7. Conclusão

As extensões ao *CosNaming* propostas neste trabalho para torná-lo tolerante a falhas em nada altera o padrão deste serviço definido pela OMG. Os serviços do *GroupPac* são inseridos de forma bastante desacoplada e transparente no *CosNamingFT*, a utilização ou não desses objetos de serviço é definido no código que implementa o objeto SI. Isto é, a utilização ou não desses serviços não implica na necessidade de qualquer modificação no código que implementa o serviço de nomes em si. Também, seguindo as recomendações da OMG, todas as comunicações entres os objetos de serviço é através do ORB (baseado no protocolo IIOP) sem o uso de qualquer mecanismo de comunicação que não esteja em conformidade com esse padrão.

As implementações apresentadas neste artigo vieram também no sentido de verificar a viabilidade de se implementar mecanismos de tolerância a falhas a nível da aplicação sobre um *middleware* CORBA, utilizando uma abordagem aberta, baseada em objetos de serviço. A abordagem implementada segue todas as recomendações da OMG no documento *Request for Proposal* (*RFP*) para tolerância a falhas, no sentido da definição de serviços e facilidades para a construção de aplicações tolerantes a falhas. As soluções propostas, dentro da linha da OMG, não devem ser dependentes de protocolos ou ferramentas proprietárias, o que diferencia esta proposta de trabalhos tais como [14] que usam uma plataforma proprietária.

O *OrbixWeb* foi usado nessas implementações mas isso também não implica na perda de transportabilidade dos códigos da aplicação para outras plataformas CORBA/Java. Além disso, a utilização do *Java* como linguagem de programação garante a portabilidade dos serviços desenvolvidos, permitindo que o mesmo conjunto de *byte-code* gerado do *CosNamingFT* pudesse ser utilizado em diferentes plataformas de máquina e sistemas operacionais como foi feito efetivamente neste trabalho. As medidas de desempenho vieram no sentido de reforçar a aplicabilidade do *CosNamingFT* como uma arquitetura para prover um serviço de nomes tolerante a falhas.

O *CosNamingFT* foi parcialmente desenvolvido dentro do contexto do projeto Sistema Nile [12], da universidade do Texas - Austin, com o objetivo de oferecer um serviço de nomes tolerante a falhas para os objetos que compõe esse sistema. A implementação deste serviço pode ser utilizado sem qualquer restrição sobre qualquer plataforma CORBA permitindo, por exemplo, que objetos CORBA desenvolvido na linguagem C++ ou outras possam também utilizar este serviço. O código compilado em Java do *CosNamingFT* pode ser obtido no seguinte endereço http://www.das.ufsc.br/~lau/CosNamingFT.html.

Bibliografia

[1] Object Management Group, **"The Common Object Request Broker 2.0/IIOP Specification"**, Revision 2.0, OMG Document 96-08-04, 1996.

[2] Object Management Group, **"CORBAservices: Common Object Services Specification"**, OMG Document March, 1997. www.omg.org.

[3] Lau C. L., J. S. Fraga, J. R. S. de Oliveira, **"Um *Framework* para Suporte a Implementação de Aplicações Tolerantes a Falhas no CORBA"**, Submetido ao Simpósio de Computadores Tolerante a Falhas – SCTF 99, SBC, Abril, 1999.

[4] Robbert V. Renesse and Kenneth P. Birman, **"Protocol Composition in Horus"** Dept. of Computer Science of the Cornell University, Mar 1995.

[5] IONA Techonologies, Ltd. **"OrbixWeb Programmer's Guide"**, 1997. www.iona.com.

[6] M. C. Little, **"Object Replication in a Distributed System"**, PhD. Thesis, University of Newcastle upon Tyne Computing Laboratory, September 1991.

[7] J. Fraga, C. Maziero, Lau L. and O. Loques, **"Implementing Replicated Services in Open Systems Using a Reflective Approach",** Proceedings of the 3th IEEE International Symposium on Autonomous Descentralized Systems - ISADS 97, Berlin - Germany, April 1997.[2]

[8] Lau C. L., J. Fraga, C. Maziero e O. Loques, **"Serviços Replicados em uma Plataforma Aberta CORBA Usando uma Abordagem Reflexiva"**, Anais do II Workshop ASAP de Sistemas Distribuídos do XIV Simpósio Brasileiro de Redes de Computadores - SBRC 96 - SBC - Fortaleza - CE, Maio 1996. [2]

[9] Lau C. L., J. Fraga, C. Maziero, **"Uma Abordagem Reflexiva Usando Suporte de Grupo para Implementar Técnicas de Replicação em Ambientes Heterogêneos"**, Anais do VII Simpósio de Computadores Tolerantes a Falhas SCTF 97 - SBC - Campina Grande, PB, Julho 1997.

[10] N. Budhiraja, K. Marzullo, F. B. Schneider and S. Toueg, **"The Primary-Backup Approach"**, Distributed Systems – Sape Mullender, Addison-Wesley, 1993.

[11] Object Management Group, **"Fault-Tolerant CORBA Using Entity Redundancy"**, RFP orbos/98-04-01, October, 1998.

[12] **Nile: National Challenge Computing Project**, www.nile.utexas.edu.

[13] A. Ricciardi and K. Birman**, "Using Process Groups to Implement Failure Detection in Asynchronous Systems"**. In Tenth ACM Symposium on the Principles of Distributed Computing . August, 1991.

[14] S. Maffeis, **"A Fault-Tolerant CORBA Name Server"**.15th IEEE Symposium on Reliable Distributed Systems, October, 1996.

[15] K. P. Birman, **"The Process Group Approach to Reliable Distributed Computing"**, Technical Report Tr91-1216, Cornell University Computer Science Department, Ithaca, N.Y., July 1991. Submitted to Comm. ACM.

[16] P. Felber, B. Garbinato and R. Guerraou,. "**The Design of a CORBA Group Communication Service**", 15th Symposium on Reliable Distributed Systems, pp 150-159, outubro de 1996.

[17] P. Maes, **"Concepts and Experiments in Computational Reflection"**, OOPSLA 87 Proceedings, pp. 147-156, October 1987.