UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Suporte a Grupo sobre a Plataforma CORBA:

Um Estudo Comparativo entre as Abordagens

Dissertação submetida à Universidade Federal de Santa Catarina como requisito para a obtenção do grau de Mestre em Engenharia Elétrica

Jorge Ricardo Souza de Oliveira

Florianópolis, março de 1999.

Suporte a Grupo sobre a Plataforma CORBA: Um Estudo Comparativo entre as Abordagens

Jorge Ricardo Souza de Oliveira

'Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em *Controle , Automação e Informática Industrial,* e aprovada em sua forma final pelo Curso de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. Joni da Silva Fraga, Dr.
Orientador

Prof. Ildemar Cassana Decker, D.Sc.
Coordenador do Curso de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Joni da Silva Fraga, Dr.
Orientador

Prof. Jean-Marie Farines, Dr.

Prof. Olinto José Varela Furtado, Dr.

A meus pais Manoel de Oliveira (*in memorian*)
e Onélia Souza de Oliveira.

A meus irmãos Heloisa Helena Souza de Oliveira,
Rafael Gustavo Souza de Oliveira
e João Paulo Souza de Oliveira.

A meus grandes amigos André Luís de Souza
e João Paulo Zacharias Pheilsticker.

Rancho de Amor à Ilha

Um pedacinho de terra
Perdido no mar!...
Num pedacinho de terra,
Belezas sem par!...
Jamais a natureza
Reuniu tanta beleza
Jamais algum poeta
Teve tanto, pra cantar!...
Num pedacinho de terra
Belezas sem par!
Ilha da moça faceira
Da velha rendeira tradicional
Ilha da velha figueira
Onde em tarde fagueira
Vou ler meu jornal
Tua lagoa formosa
Ternura de rosa
Poema ao luar
Cristal onde a lua vaidosa
Sestrosa, dengosa
Vem se espelhar.

Cláudio Alvin Barbosa (Zininho)

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus que sempre tem me acompanhado e guiado durante a vida. Além disso, agradeço ao Prof. Joni da Silva e Fraga pelo seu empenho e incentivo durante a orientação e ao amigo Lau Cheuk Lung pelas conversas e contribuições que tanto enriqueceram este trabalho.

Agradeço também a todos os professores, colegas e funcionários do Programa de Pós-graduação em Engenharia Elétrica que de alguma forma, ou de outra, contribuíram para a realização deste trabalho. Em especial, agradeço a meus amigos: Alexandre, Carlos, César, Cristian, Eládio, Howard, Ivana, Karina, Marcos, Montez, Pierre e Sandro pela convivência e apoio durante o período do curso de mestrado.

Finalmente, agradeço a meus pais, irmãos e amigos que sempre me incentivaram e apoiaram.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

O padrão CORBA une as tecnologias de orientação a objetos e computação distribuída, oferecendo uma infra-estrutura que permite a comunicação entre objetos remotos. Todavia, este padrão não apresenta mecanismos de tolerância a faltas. Neste sentido, este trabalho visa apresentar e discutir extensões do padrão CORBA que fornecem suporte ao processamento de grupos de objetos. O processamento de objetos em grupo permite que a disponibilidade e confiabilidade de uma aplicação sejam incrementadas por meio da replicação de seus objetos.

Na literatura são identificadas três abordagens para a adição de suporte a grupo sobre ORBs CORBA: integração, serviço e interceptação. Cada uma destas abordagens apresenta características que procuram explorar aspectos de desempenho, transparência, conformidade com padrões abertos e facilidade de uso. Na abordagem de integração [Isis93, Maffeis95b], o núcleo do ORB é alterado para permitir a execução dos mecanismos de processamento de grupo. Na abordagem de serviço [Felber97, Liang98], estes mecanismos são disponibilizados como um serviço CORBA [OMG95b]. Finalmente, na abordagem de interceptação [Liang96, Narasimhan97a], o processamento de grupo é ativado transparentemente por meio de mecanismos interceptores.

No decorrer deste trabalho é descrito um estudo comparativo das abordagens de integração, serviço e interceptação. Neste sentido, além de discussões sobre ambientes de suporte a grupo sobre ORBs CORBA existentes, são desenvolvidas implementações sobre as quais são realizadas análises de desempenho.

ABSTRACT

The CORBA standard joins object orientation and distributed computing technologies, supporting a framework to allow communication between remote objets. However, this standard does not address fault-tolerance mechanisms. In this context, this work aims to present and comment extensions to the CORBA standard that support object group processing. Object group processing allows increment the availability and reliability of applications by replicating their objects.

In the literature, three approaches to add group support over CORBA ORBS are identified: integration, service and interception. Each of these approaches has characteristics that aim to explore performance, transparence, open standard accordance and easy use aspects. In the integration approach [Isis93, Maffeis95b], the ORB core is changed to allow the execution of group processing mechanisms. In the service approach [Felber97, Liang98], these mechanisms are made available as a CORBA service [OMG95b]. Finally, in the interception approach [Liang96, Narasimhan97a], the group processing is activated transparently by interceptor mechanisms.

In this work, a comparative study among the integration, service and interception approaches is described. In this context, besides discussions over existent environments to support groups over CORBA ORBs, implementations are developed and performance analyses over them are done.

1. INTRODUÇÃO

Os sistemas distribuídos possuem uma série de vantagens sobre os sistemas centralizados tradicionais, como maior desempenho, flexibilidade e confiabilidade. Além disso, a computação distribuída possui uma motivação econômica, pois possibilita o compartilhamento de recursos e o conseqüente aumento da utilização destes. Entretanto, tais sistemas possuem uma complexidade superior ao antigo modelo de processamento centralizado.

A fim de diminuir o problema de complexidade dos sistemas distribuídos, surgiram várias propostas para facilitar a sua implementação. Todavia, a maioria das soluções é proprietária e, por isto, a integração de diferentes propostas se torna muito difícil. Com o intuito de superar este problema de integração, foram desenvolvidos vários padrões abertos para sistemas distribuídos, como o ODP[1] da ISO/CCITT [ISO93], a arquitetura ANSA [Oskiewicz93], o DCE/OSF[2] [Rosenberry92] e o CORBA[3] [OMG95a]. Dentre estes padrões, o CORBA é o que merece maior destaque, alcançando grande aceitação entre os produtores e usuários de *software*.

O padrão CORBA une as tecnologias de orientação a objetos e computação distribuída, consistindo em uma infra-estrutura para aplicações distribuídas orientadas a objeto. Seu objetivo é permitir a comunicação entre objetos remotos, possivelmente implementados em diferentes plataformas de *hardware* e *software*. Atualmente, o padrão CORBA oferece apenas serviços básicos como os mecanismos para comunicação. Funções mais específicas, como o provimento suporte a tolerância a faltas e outras, precisam ser implementadas e adicionadas as especificações CORBA.

[1] *ODP vem do inglês: "Open Distributed Processing".*
[2] *DCE vem do inglês: "Distributed Computing Environment".*
[3] *CORBA vem do inglês: "Common Object Request Broker Architecture".*

O processamento de objetos em grupo pode fornecer suporte a aplicações que apresentam requisitos de consistência de estado entre as réplicas, alta disponibilidade, tolerância a faltas, balanceamento de carga ou suporte a trabalho cooperativo. Neste contexto, o uso de mecanismos de processamento de grupo permite, entre outros, que a disponibilidade e a confiabilidade de uma aplicação possam ser incrementadas por meio da replicação de seus objetos. Com o intuito de aproveitar estas características de tolerância a faltas, foram desenvolvidos vários protótipos ou mesmo produtos de *middlewares*[1] CORBA com suporte a grupos de objetos. A OMG ainda não tem especificado um serviço de suporte ao processamento de grupo ou de tolerância a faltas no CORBA. Existem somente esforços em resposta a um primeiro documento *Request For Proposal (RFP)* requisitando propostas para a padronização de funcionalidades CORBA de suporte a aplicações tolerantes a faltas [OMG98b]. Os requisitos estipulados neste documento prevêem o fornecimento de serviços e facilidades para a construção de aplicações confiáveis. Estas propostas devem apresentar soluções abertas, não dependentes de protocolos ou ferramentas proprietárias.

Este trabalho tem por objetivo a apresentação e discussão de soluções para a adição de mecanismos de processamento de objetos em grupo ao padrão CORBA. Para tal foram identificadas três abordagens:

- **Abordagem de Integração** – A abordagem de integração [Isis93, Maffeis95b] consiste na construção ou na modificação de um *middleware* CORBA existente, adicionando mecanismos de processamento em grupo. Nesta abordagem, o núcleo do ORB é alterado para que as aplicações não possam distinguir objetos simples de grupos de objetos, oferecendo um alto grau de transparência.
- **Abordagem de Serviço** – A abordagem de serviço [Felber97, Liang98] consiste em prover os mecanismos de processamento em grupo encapsulados na forma de um serviço CORBA (COSS[2]). O ORB permanece inalterado, não conhecendo nada a respeito de grupos. Esta abordagem é similar a adotada para aprimorar o CORBA com

[1] *O termo inglês "middleware" é utilizado para designar camadas de software acima do sistema operacional que ofereçam algum serviço de suporte às aplicações.*
[2] *COSS vem da expressão em inglês: "Common Object Service Specification".*

persistência, transações, canais de eventos e outros serviços comuns utilizados pelas aplicações [OMG95b].

- **Abordagem de Interceptação** – A abordagem de interceptação [Liang96, Narasimhan97a] oferece o suporte a grupos de objetos por meio do uso de mecanismos de filtros ou interceptores. Neste contexto, as chamadas aos objetos CORBA são capturadas e o fluxo de controle é desviado de forma transparente para um suporte que implementa a comunicação em grupo.

No decorrer deste trabalho é descrito um estudo comparativo das abordagens de integração, serviço e interceptação, levantando questões como desempenho, transparência, conformidade com o padrão CORBA, flexibilidade e facilidade de uso. Neste sentido, além de discussões a respeito de ambientes de suporte a grupo sobre ORBs CORBA já existentes, são desenvolvidas implementações, sobre as quais são realizadas análises de desempenho. O estudo desenvolvido neste trabalho faz parte do projeto GROUPAC, que tem a finalidade de desenvolver um grupo de objetos de serviço para o suporte a aplicações confiáveis [Lung99].

Este trabalho está dividido da seguinte maneira. No capítulo 2, o padrão CORBA é apresentado em mais detalhes, descrevendo-se os componentes que o formam. No capítulo 3 são estudados conceitos de processamento de grupos de objetos em sistemas distribuídos, apresentando-se áreas de utilização e escolhas de implementação. Nos capítulos 4, 5 e 6 são detalhadas as abordagens de integração, de serviço e de interceptação para o suporte a grupos em ORBs CORBA. No capítulo 7 é feita uma comparação entre as abordagens de acordo com uma série de critérios estabelecidos, além disso são apresentadas as implementações desenvolvidas e as análises de desempenho. Finalmente no capítulos 8 são feitas conclusões sobre as questões apresentadas e levantadas perspectivas futuras para este trabalho.

2. PADRÃO CORBA

2.1 Introdução

A *Object Management Group* (OMG) é uma organização internacional mantida por mais de 750 membros, incluindo produtores, usuários e vendedores de *software*, que visa promover a teoria e a prática da tecnologia de orientação a objetos no desenvolvimento de *software*. Para tal, a sua área de atuação inclui o estabelecimento de normas para a indústria e de especificações de orientação a objetos. A OMG não produz *software*, apenas cria padrões que podem ser seguidos. Estes padrões têm o objetivo de aumentar a reusabilidade, portabilidade e interoperabilidade de *softwares* orientados a objetos em ambientes distribuídos heterogêneos [Keahey98, OMG95b].

A *Object Management Architecture* (OMA) desenvolvida pela OMG, pretende disciplinar a construção de um ambiente para aplicações heterogêneas sobre as principais plataformas de *hardware* e sistema operacional disponíveis. O *Common Object Request Broker Architecture* (CORBA) é parte fundamental da arquitetura OMA, definindo os meios pelos quais os objetos podem se comunicar [OMG95b, OMG97, Schmidt97].

Neste capítulo primeiro é discutida a arquitetura OMA, expondo os seus componentes. Então, é detalhado o padrão CORBA, apresentando uma conceituação geral e os elementos que o formam.

2.2. Arquitetura OMA

A arquitetura OMA provê uma infra-estrutura conceitual sobre a qual todas as especificações da OMG são baseadas. Conforme ilustrado na figura 2.1, esta arquitetura é composta por cinco componentes [OMG95b, OMG97, Schmidt97]:

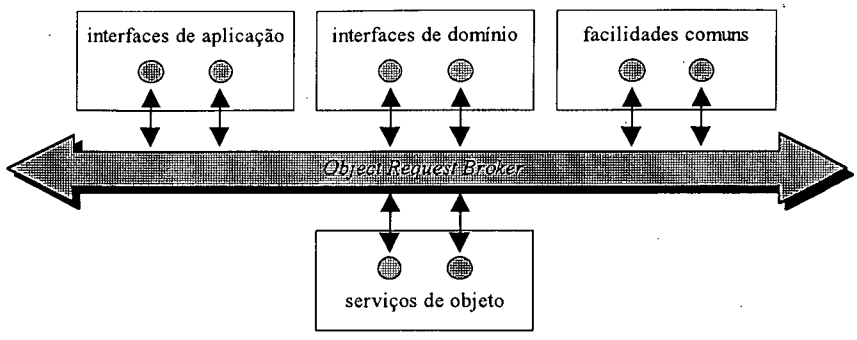


Figura 2.1 – Componentes da arquitetura OMA

- ***Object Request Broker* (ORB)** – O ORB é o núcleo da arquitetura OMA e eqüivale a arquitetura CORBA propriamente dita. O ORB provê uma infra-estrutura para comunicação que permite aos objetos conversarem entre si independente da plataforma e das técnicas usadas para implementá-los.
- **Serviços de Objetos** – Os serviços de objeto ou serviços CORBA são uma coleção de interfaces que provêem funções básicas para facilitar a implementação e utilização dos objetos de aplicação. Os serviços são independentes do domínio das aplicações e fundamentais para o desenvolvimento destas. Vários serviços foram projetados e adotados como padrões pela OMG. Entre estes estão o serviço de ciclo de vida usado para a criação e exclusão de objetos, o serviço de persistência para armazenar os objetos de forma persistente e o serviço de transações que permite a objetos distribuídos participarem de transações atômicas. Em conjunto com o ORB, os serviços de objeto estão relacionados a aspectos de infra-estrutura do sistema distribuído.
- **Facilidades Comuns** – As facilidades comuns ou facilidades CORBA são uma coleção de interfaces que pode ser compartilhada por várias aplicações, entretanto que não são tão fundamentais quanto os serviços de objeto. Por exemplo, mecanismos para a gerência de sistemas e correio eletrônico podem ser classificadas como facilidades comuns. Assim como os serviços de objeto, as facilidades comuns podem ser usadas por aplicações de qualquer domínio, entretanto, as facilidades comuns oferecem funções mais próximas aos usuários.
- **Interfaces de Domínio** – As interfaces de domínio são uma coleção de interfaces orientadas a domínios de aplicação específicos. Por exemplo, interfaces orientadas a domínios como telecomunicações, medicina ou finanças. Diferente dos serviços de

objeto e das facilidade comuns, que são dirigidos a todas as aplicações, as interfaces de domínio são voltadas a apenas aplicações de um certo domínio.

- **Interfaces de Aplicação** – As interfaces de aplicação são interfaces desenvolvidas especificamente para uma dada aplicação. Estas interfaces consistem na aplicação propriamente dita, sendo a camada superior da arquitetura OMA. Desta forma, estas interfaces não são padronizadas pela OMG.

2.3. Arquitetura CORBA

O padrão CORBA consiste em uma arquitetura orientada a objetos para sistemas distribuídos heterogêneos. O CORBA provê uma infra-estrutura que permite aos objetos se comunicarem independente de sua plataforma ou das técnicas usadas na sua implementação. Esta arquitetura define os objetos como as unidades básicas de distribuição, sendo que cada objeto pode suportar uma coleção de operações por meio de uma interface de componente. Esta interface é especificada em uma linguagem de definição de interface (IDL[1]), que é independente da linguagem de implementação dos objetos [OMG95a].

Por meio do provimento de uma arquitetura orientada a objetos com herança de interface, interoperabilidade entre ORBs e independência de plataforma, o CORBA alcança alguns dos requisitos mais essenciais da computação moderna, que é maximizar a portabilidade, reusabilidade e interoperabilidade de *software*. Neste contexto, é possível utilizar uma mesma aplicação com diferentes *middlewares* CORBA, apenas a recompilando. Além disso, aplicações em ORBs distintos podem cooperar entre si [OMG95a].

Objetos clássicos são acessíveis somente em um único programa e o mundo exterior não conhece nada a respeito deles. De forma inversa, os objetos CORBA são acessíveis em qualquer ponto da rede. Estes objetos podem ser acessados por clientes locais ou remotos por meio da invocação de suas operações. A linguagem utilizada para implementar os objetos servidores e o sistema operacional onde eles estão sendo executados são totalmente transparentes para os clientes, que precisam conhecer apenas a interface publicada pelos

[1] *IDL vem do inglês: "Interface Definition Language". Apesar de existirem linguagens IDL definidas por outras organizações além da OMG, neste trabalho é utilizada a sigla IDL para identificar apenas a OMG IDL.*

servidores. Os clientes utilizam referências de objetos para identificar objetos remotos. As referências são válidas em todo o sistema e podem ser passadas de um nodo para outro. Além disso, os objetos não são restritos aos papéis de cliente ou servidor, eles podem atuar algumas vezes como cliente e outras como servidor [Keahey98, OMG95a].

A figura 2.2 ilustra os componentes da arquitetura CORBA [Keahey98, OMG95a, Schmidt97]. A descrição destes componentes é feita a seguir:

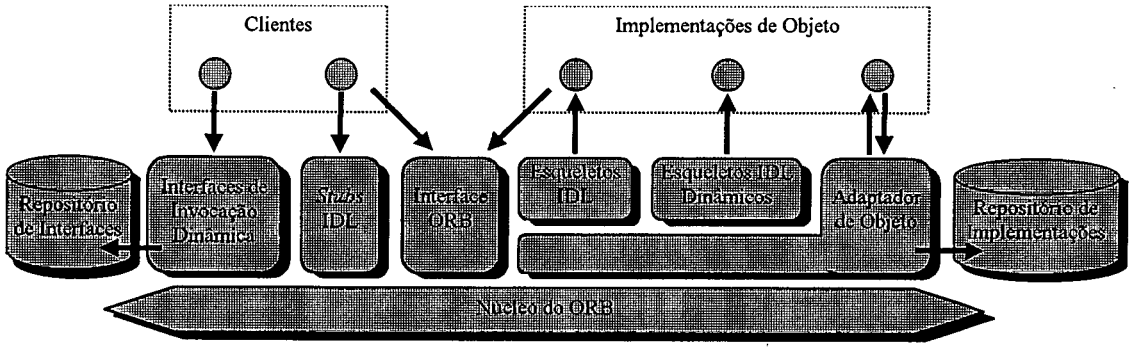


Figura 2.2 – Componentes da arquitetura CORBA

- **Implementações de Objeto** – As implementações de objeto ou servidores definem as operações que implementam as interfaces IDL do CORBA. Estas implementações podem ser escritas em diversas linguagens como: *C*, *C++*, *Java*, *Smalltalk*, *Ada* e outras. A plataforma CORBA utilizada deve permitir o porte da linguagem escolhida.
- **Clientes** – Os clientes são as entidades que invocam as operações das implementações de objeto. O acesso aos serviços de um objeto remoto deve ser transparente para os clientes. Este acesso deve ser tão simples quanto uma chamada a um método de um objeto local.
- **Núcleo do ORB** – O núcleo do ORB provê mecanismos para comunicar de forma transparente as requisições de um cliente para a implementação de objeto adequada. Quando um cliente invoca uma operação em um servidor, o núcleo do ORB, em conjunto com outras estruturas, é responsável por encontrar a implementação de objeto adequada, ativá-la de forma transparente e retornar uma resposta ao cliente.
- **Interface ORB** – O núcleo do ORB é uma entidade lógica que pode ser implementada de várias formas, como uma coleção de processos ou bibliotecas. Para separar as aplicações dos detalhes de implementação, a especificação CORBA definiu uma interface abstrata para o ORB. Esta interface provê várias funções de auxílio, como

converter referências de objetos para *strings* e vice-versa ou criar uma lista de argumentos para requisições feitas por meio de uma interface de invocação dinâmica.

- ***Stubs* IDL** – Os *stubs* IDL[1] descrevem os serviços fornecidos pelas implementações de objeto. Estas estruturas provêem uma interface aos clientes para o acesso a tais serviços. Neste contexto, os *stubs* são responsáveis pelo transporte das requisições dos clientes até o núcleo do ORB, que é então encarregado de localizar a implementação de objeto adequada e ativá-la.
- **Esqueletos IDL** – Os esqueletos IDL estáticos provêem uma interface ao núcleo do ORB para o acesso às implementações de objeto. Em conjunto, os *stubs* e os esqueletos IDL permitem a comunicação entre os clientes e as implementações de objeto através do núcleo do ORB.
- **Interfaces de Invocação Dinâmica** – As interfaces de invocação dinâmica permitem aos clientes acessarem diretamente os mecanismos de requisição providos pelo núcleo do ORB. As aplicações podem utilizar estas interfaces para emitir requisições a objetos, sem necessitar que *stubs* sejam ligados a eles. Ao contrário dos *stubs* que permitem apenas requisições do tipo RPC[2], as interfaces de invocação dinâmica permitem chamadas não bloqueantes e chamadas sem confirmação[3].
- **Esqueletos IDL Dinâmicos** – Os esqueletos IDL dinâmicos são análogos as interfaces de invocação dinâmica. Estes esqueletos permitem ao núcleo do ORB entregar requisições a uma implementação de objeto que em tempo de compilação não tem conhecimento do tipo de objeto que está implementando. O cliente que faz uma requisição não tem idéia se a implementação está utilizando esqueletos IDL dinâmicos ou estáticos.
- **Adaptador de Objeto** – O adaptador de objeto auxilia o núcleo do ORB na entrega de requisições e na ativação de objetos. Um adaptador de objeto associa as implementações de objeto com o ORB, permitindo a estas implementações acessarem serviços do ORB. Em particular, a especificação CORBA define um adaptador de

[1] *Os stubs IDL também podem ser referenciados como interfaces de invocação estática.*
[2] *A sigla RPC identifica uma chamada de procedimento remoto e vêm da expressão em inglês: "Remote Procedure Call". As chamadas do tipo RPC são bloqueantes, ou seja, o seu invocador é interrompido até o recebimento da resposta.*
[3] *As chamadas não bloqueantes possuem as primitivas de envio e resposta separadas. As chamadas sem confirmação necessitam apenas da primitiva de envio.*

objetos básico (BOA[1]) que oferece funcionalidades padronizadas. Todavia, é permitida a criação de adaptadores de objeto especializados que podem prover funções como o suporte a banco de dados, persistência, acesso a bibliotecas de objetos e outros.

- **Repositório de Interfaces** – As interfaces de objeto, especificadas em IDL, podem ser adicionadas ao repositório de interfaces. As interfaces de invocação dinâmica permitem ao cliente especificar requisições para objetos cujas definições não são conhecidas em tempo de compilação. Para tal, o cliente tem que criar uma requisição que deve incluir a referência para o objeto, a operação e a lista de parâmetros. A identificação do objeto e da operação é recuperada do repositório de interfaces, que é um banco de dados que fornece armazenamento persistente de definições de interface de objetos. O repositório de interfaces também pode conter informações de tipos de parâmetros, depuração e outros.
- **Repositório de Implementações** – O repositório de implementações provê em tempo de execução informações sobre as implementações de objeto, como: as classes suportadas, os objetos instanciados e seus identificadores, informações administrativas e de depuração. Estas informações são utilizadas pelo adaptador de objetos e o núcleo do ORB para localizar e ativar as implementações de objetos.

2.4 Conclusões do Capítulo

Neste capítulo é feita uma revisão de vários conceitos a respeito do padrão CORBA. Tal revisão, além de situar o padrão CORBA dentro da arquitetura OMA proposta pela OMG, visa estabelecer definições e um vocabulário padrão que são utilizados no restante deste trabalho, unificando várias informações encontradas na literatura.

Em particular, este capítulo não pretende esgotar as discussões sobre o padrão CORBA, uma vez que este assunto, além de extenso, vem sofrendo constantes aprimoramentos. Posteriormente, no decorrer deste trabalho, outros conceitos CORBA são estudados em mais detalhes. No capítulo 6, é ilustrada a utilização prática de interfaces de invocação e esqueletos dinâmicos. No capítulo 7 são apresentados os protocolos GIOP e IIOP utilizados nas comunicações via ORB, além dos mecanismos interceptores.

[1] *A sigla BOA vêm do inglês: "Basic Object Adapter".*

3. PROCESSAMENTO EM GRUPO

3.1 Introdução

Um grupo consiste em uma associação em que seus membros apresentam uma relação abstrata comum, semânticas de aplicação comuns ou políticas internas comuns [Lea94]. Geralmente, os membros do grupo podem ser vistos como uma entidade lógica única. Neste contexto, o número e a localização exata destes membros, além dos protocolos comunicação utilizados dentro do grupo devem ser transparentes [Hagsand92, Isis93, Lea94, Liang90, Maffeis93].

As motivações para a inclusão do processamento em grupo nos modelos de programação distribuída são muitas. Em particular, o processamento de grupo pode ser usado em sistemas distribuídos para aumentar a disponibilidade de recursos compartilhados ou no processamento replicado por razões de tolerância a faltas. Além disso, o processamento de grupo pode ser utilizado para dar suporte a aplicações de trabalho cooperativo, facilitando as interações nestas aplicações, ou ainda, no balanceamento de carga para aumentar o desempenho de um serviço prestado.

As tecnologias de orientação a objetos e processamento em grupo foram dois domínios de pesquisa distintos até alguns anos atrás. Neste contexto, o conceito de grupo, por si só, não tem nenhuma relação com a orientação a objetos. Os membros de um grupo podem ser objetos, processos, estruturas de dados ou outros. Todavia, atualmente tem surgido um grande número de produtos e trabalhos de pesquisa onde estas tecnologias estão integradas [Lea94, Liang90]. A orientação a objetos fornece um alto nível de abstração para descrever os componentes de *software*. Além disso, os mecanismos de herança de classe e encapsulamento facilitam a portabilidade e reusabilidade de *software*. Desta forma, o processamento de objetos em grupo oferece uma infra-estrutura poderosa e

flexível que une os benefícios da orientação a objetos e comunicação em grupo, podendo ser usado para diversas finalidades [Birman91, Isis93, Maffeis93].

Atualmente, vários sistemas, orientados a objetos ou não, fazem uso da noção de grupo para suportar aplicações distribuídas, como o Horus [Renesse93a], Isis [Birman90], Totem [Amir95] e Transis [Amir92]. Além disso, já existem várias propostas e implementações que estendem o padrão CORBA com processamento em grupo, como o OGS [Felber97], Eternal [Narasimhan97b], Orbix+Isis [Isis95] e Electra [Maffeis96].

A seguir, o conceito de grupo e suas áreas de utilização são detalhados. Para tal, é apresentada uma classificação de tipos de grupo [Liang90]. Posteriormente, é discutida a difusão de mensagens em grupo, relacionando-se as várias formas de ordenar-se as mensagens recebidas por um grupo. Então, são apresentados mecanismos de gestão de grupo como a pertinência[1] e a transferência de estado. Finalmente, são detalhadas as diversas formas de transparência de grupo, levando em conta o modelo cliente-servidor.

3.2. Tipos de Grupo

Várias classificações de tipos de grupo, utilizando diversos critérios, têm sido apresentadas na literatura [Birman93, Kaashoek92, Liang90]. Neste trabalho é apresentada a classificação de grupos de acordo com a estrutura de seus componentes [Liang90, Nacamura96]. Segundo esta classificação, os membros do grupo implementam um serviço a partir de um conjunto de operações e dados. A descrição informal dos tipos de grupo previstos por esta classificação é feita a seguir:

- **Grupo DOH**[2] – Todos os membros de um grupo DOH compartilham, na execução do serviço, os mesmos dados e operações. Os grupos DOH são usados principalmente para aumentar a confiabilidade e a disponibilidade de serviços.
- **Grupo OHO**[3] – Nos grupos OHO, o espaço de dados é particionado entre os membros do grupo. Todavia, cada membro suporta o mesmo conjunto de operações. Este grupo é usado principalmente para distribuir a carga entre seus membros.

[1] *A pertinência de grupo é normalmente referenciada em inglês como: "group membership".*
[2] *A sigla DOH vem da expressão em inglês "Data and Operation Homogeneous".*
[3] A sigla OHO vem da expressão em inglês "*Operation Homogeneous Only*".

- **Grupo DHO**[1] – Os membros de um grupo DHO compartilham os dados, todavia as operações são diferentes. Estes grupos são chamados de grupos funcionalmente distribuídos [Oskiewicz93], sendo utilizados em aplicações cooperativas.
- **Grupo Het**[2] – Em um grupo Het tanto os dados como as operações são diferentes. Neste tipo de grupo pode haver ou não cooperação entre os membros. A associação os membros é principalmente para facilitar o fluxo de informações.

Neste trabalho, são estudados principalmente os grupos DOH e DHO. Estes tipos de grupo são os mais apropriados para suportar servidores tolerantes a faltas.

3.3. Comunicação de Grupo

A noção de grupo está diretamente associada com a disseminação de mensagens entre os membros do grupo. Esta disseminação é realizada usando mecanismos de comunicação de grupo. Na literatura diferentes algoritmos de comunicação de grupo são normalmente identificados como protocolos de difusão[3]. Em alguns dos tipos de grupos citados anteriormente, é possível o uso de difusão não confiável, onde não é exigido que todos os membros do grupo recebam a mensagem enviada [Nacamura96].

Neste trabalho, o conceito de grupo é utilizado para dar suporte a aplicações tolerantes a faltas em sistemas distribuídos. Desta forma, difusões não confiáveis não são adequadas devido a inconsistências que poderiam ser geradas nos diferentes membros do grupo. Os protocolos de difusão confiável, por sua vez, suportam serviços de envio de mensagens que garantem, mesmo em presença de falhas, um comportamento bem definido em relação às mensagens difundidas no grupo. Para tal, estes protocolo são fundamentados em três propriedades básicas [Cristian95, Powell91]: acordo, ordenação e terminação[4]. Neste contexto, uma mensagem só é considerada aceita em um participante do grupo se

[1] A sigla DHO vem da expressão em inglês "*Data Homogeneous Only*".
[2] A sigla Het vem da expressão em inglês "*Heterogeneous*".
[3] O termo difusão é normalmente referenciado em inglês como "*broadcast*" ou "*multicast*".
[4] Conforme o maior ou menor rigor das propriedades de acordo, ordenação e terminação são identificadas diferentes categorias de protocolos de difusão de grupo.

estas propriedades são atendidas. O ato de aceitação da mensagem é chamado na literatura de engajamento da mensagem[1].

A propriedade de acordo deve garantir em presença de falhas, que ou todos os participantes corretos engajam a mensagem difundida ou que todos eles desconsideram esta mensagem. Quanto mais restritivas forem as hipóteses de faltas, menos complexos e de menor custo são os algoritmos que atendem a propriedade de acordo. A propriedade de terminação garante que o protocolo de difusão confiável termine. Finalmente, a propriedade de ordenação garante que todas as mensagens difundidas em um grupo estarão sujeitas a uma ordem de engajamento. A seguir são descritos vários tipos de ordenação [Shrivastava92]:

- **Ordenação por Ordem de Chegada** – A ordenação por ordem de chegada ou ordenação FIFO[2] assegura que as mensagens de um mesmo emissor, são engajadas pelos participantes do grupo segundo a ordem de emissão. Mensagens de emissores diferentes podem ser recebidas em membros distintos do grupo em ordens opostas.
- **Ordenação Causal** – A ordenação causal é uma extensão natural da ordenação FIFO considerando-se causalidade potencial. Segundo este tipo de ordenação, caso uma mensagem seja a causa potencial de outra, ela deve ser engajada pelos participantes do grupo primeiro. Mensagens que não apresentam relação de causa potencial podem ser engajadas em qualquer ordem. A ordenação causal atende a ordens semânticas das aplicações.
- **Ordenação Total** – A ordenação total garante que todas as mensagens são engajadas numa mesma ordem por todos os participantes do grupo. A ordem em que todos os participantes devem engajar as mensagens envolve mensagens de diferentes emissores e não necessariamente corresponde a ordem cronológica de emissão das mensagens. Na literatura, este tipo de ordenação é chamado de ordenação atômica.

Segundo esta classificação, a ordem causal inclui a ordem FIFO, e a ordem total é uma propriedade independente.

[1] O termo engajamento de mensagem é normalmente referenciado em inglês como: "*message commit*".
[2] A sigla FIFO vem da expressão em inglês: "*First In, First Out*".

3.4. Gestão de Grupo

A gestão de grupo visa assegurar o correto funcionamento do grupo. Para tal, são oferecidos mecanismos para monitorar e controlar a entrada e saída de membros em um grupo. Além disso, são fornecidos os meios para manter os estados dos membros do grupo. A seguir são relacionados vários mecanismos para a gestão de grupo.

A pertinência de grupo tem o objetivo de garantir que todos os membros de um grupo tenham a mesma idéia a respeito da composição deste grupo [Birman91, Isis93]. A composição de um grupo consiste na lista dos membros correntes deste grupo e é também chamada de visão. Desta forma, quando um membro junta-se ou deixa o grupo, ocorre uma mudança de pertinência e uma nova visão deve ser disponibilizada aos membros atuais do grupo. Estas mudanças de pertinência devem ser notificadas na mesma ordem a todos os membros do grupo, garantindo que estes mantenham informações de pertinência consistentes [Lea94].

No caso de grupos dos tipos DOH e DHO, os membros do grupo compartilham um estado comum. Portanto, quando um novo membro junta-se a um grupo, ele tem que receber o seu estado atual. Isto é feito por meio de mecanismos de transferência de estado, que transparentemente transmitem o estado dos membros correntes para o novo membro. Neste contexto, cada membro de grupo precisa suportar operações para receber e enviar o estado compartilhado.

Os membros podem deixar um grupo voluntariamente ou podem falhar e terem que ser excluídos do grupo. Para detectar e tratar a ocorrência de falhas em cada membro podem ser utilizados mecanismos de temporização e notificações de funcionamento[1]. Neste contexto, os membros que periodicamente não enviarem qualquer mensagem ao grupo serão considerados falhos e então desligados do grupo [Isis93]. Em alguns casos, para permitir que um grupo seja recuperado, mesmo na situação de falha de todos os seus membros, o estado atual do grupo e as requisições que ainda não foram tratadas podem eventualmente ser armazenados em disco. Para tal, podem ser utilizados pontos de recuperação[2]. Desta forma, cada vez que um ponto de recuperação é alcançado pelos

[1] As notificação de funcionamento são normalmente referenciados em inglês como: "*keepalives*".
[2] Os pontos de recuperação são normalmente referenciados em inglês como: "*checkpoints*".

membros de um grupo, o seu estado atual e as requisições subseqüentes ao grupo devem ser copiados em disco.

3.5. Transparência de Grupo

O processamento de grupo é muitas vezes utilizado segundo um modelo cliente-servidor. Desta forma, os grupos podem agir como fornecedores ou usuários de um serviço. Neste contexto, é interessante que um grupo possa ser visto como um cliente ou um servidor qualquer, isto é, os grupos devem ser transparentes. Em particular, existem três tipos de transparências de grupo: transparências de cliente, de servidor e de membro de grupo [Isis93].

A transparência de cliente garante que os clientes enviem requisições e recebam respostas de um grupo servidor como se este fosse um servidor único. Neste contexto, um servidor simples deve poder ser facilmente substituído por um grupo, obtendo-se alta disponibilidade, sem que seja necessário alterar os clientes. Normalmente, os clientes recebem apenas uma única resposta do grupo servidor. Desta forma, devem ser providos mecanismos para coletar os resultados de todos os servidores, garantindo o envio de apenas uma resposta. Em situações onde o processamento de grupo é utilizada apenas para aumentar a disponibilidade de um serviço, estes mecanismos podem apenas enviar o resultado do primeiro servidor a processar a requisição, descartando os demais. Todavia, caso os servidores possam produzir respostas distintas, os resultados podem ser comparados, sendo enviada ao cliente a resposta mais freqüente [Isis93, Liang90]. Além disso, em alguns casos, as requisições dos clientes não precisam ser executadas por todos os servidores. Por exemplo, apenas um membro do grupo pode executar uma operação de leitura, ou ainda, um membro pode executar operações de escrita e retornar os resultados tanto ao cliente quanto aos demais servidores[Isis93].

Em situações especiais, onde o cliente tem consciência da existência do grupo, toda a coleção de respostas dos servidores pode ser retornada ao cliente. Nestes casos, é de responsabilidade do cliente determinar a resposta correta, podendo utilizar algoritmos mais elaborados para tal, como a utilização de ajustadores. Pode-se perceber que com esta

abordagem, o grupo não é totalmente transparente para o cliente. Ou seja, embora o cliente invoque o grupo como um servidor único, ele recebe várias respostas [Isis93].

A transparência de servidor garante que um grupo cliente possa interagir com um servidor como um cliente único o faria. Neste contexto, quando um servidor recebe uma requisição, ele não precisa saber se o cliente é uma entidade simples ou um grupo. O suporte a transparência de servidor depende da organização interna do grupo. Em alguns casos, os algoritmos internos do grupo garantem que somente um membro envie a requisição ao servidor. Em outros casos, cada membro envia uma requisição idêntica. Nesta última abordagem, estas requisições devem ser agrupadas em uma única requisição no servidor. Entretanto, independente da abordagem utilizada, o resultado obtido pelo servidor deve ser disseminado a todos os membros do grupo cliente [Isis93].

A transparência de membro de grupo garante que os membros possam se comportar como se não fizessem parte de um grupo. Desta forma, os membros não podem diferenciar comunicações individuais de mensagens enviadas a todo o grupo. Neste caso, os protocolos de gestão de grupo devem ser transparentes para os membros do grupo.

3.5 Conclusões do Capítulo

Como é dito anteriormente, o processamento em grupo tem sido amplamente discutido na literatura. Todavia, tais discussões apresentam, muitas vezes, conceitos divergentes. Em particular, existem trabalhos na literatura que consideram apenas grupos de processos ou apenas grupos de objetos. Além disso, enquanto alguns trabalhos permitem que entidades distintas façam parte de um grupo, outros exigem que os membros de um grupo sejam réplicas exatas.

Neste capítulo, busca-se apresentar uma visão mais geral do conceito de grupo, abrangendo várias definições da literatura. Definições mais específicas, propostas por outros autores, podem muitas vezes, ser consideradas como casos particulares daquilo que é exposto aqui. Todavia, como este trabalho visa explorar o processamento em grupo com o intuito de aumentar a confiabilidade em aplicações orientadas a objeto, os comentários realizados tratam com mais atenção desta abordagem.

4. ABORDAGEM DE INTEGRAÇÃO

4.1 Introdução

A abordagem de integração consiste na construção ou na modificação de um *middleware* CORBA existente, adicionando mecanismos de processamento de grupo. Nesta abordagem, o *middleware* CORBA é alterado para que as aplicações não possam distinguir objetos simples de grupos de objetos, oferecendo um alto grau de transparência. A idéia principal desta abordagem é que o processamento de grupo é suportado por um sistema de comunicação abaixo do núcleo do ORB. Todas as chamadas que envolvam comunicação ou gestão de grupo são repassadas pelo núcleo do ORB a este suporte de mais baixo nível. Esta abordagem já foi adotada em algumas plataformas CORBA existentes, como o Orbix+Isis e o Electra.

Na abordagem de integração, as referências de objeto passam a poder identificar tanto um objeto único, como um grupo de objetos. O ORB é responsável por distinguir estas referências. A figura 4.1 ilustra a execução de uma requisição em um grupo de objetos. No lado cliente, a requisição quando passada ao ORB, é reconhecida pelo mesmo como uma requisição endereçada a um grupo, sendo convertida em uma chamada de difusão na ferramenta de mais baixo nível que dá suporte à comunicação em grupo. Em seguida, a operação adequada é invocada pelo ORB em cada membro do grupo de objetos servidores. Os resultados do processamento retornam pelo mesmo caminho, porém no sentido inverso. Finalmente, as respostas são coletadas no lado cliente, submetidas a alguma função de consenso e retornadas ao objeto que fez a requisição [Felber96].

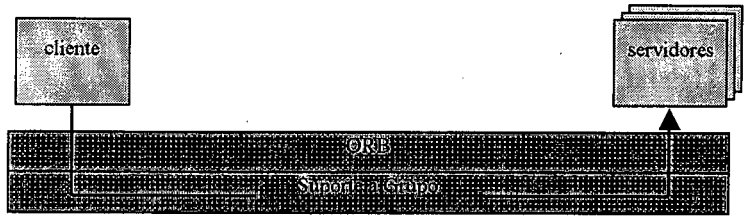


Figura 4.1 – Execução de uma requisição na abordagem de integração

As implementações da abordagem de integração podem fazer uso de adaptadores de objetos disponíveis que fornecem acesso a facilidades para a gestão de grupo. Esta possibilidade é permitida pela especificação CORBA. Todavia, nos *middlewares* Orbix+Isis e Electra, o próprio adaptador básico é estendido, permitindo que todos os objetos do sistema possam tornar-se membros de grupos.

A seguir são descritos vários *middlewares* que implementam a abordagem de integração. Primeiro é apresentado o Isis/C++ [Hagsand91]. Embora este *middleware*, diferente dos demais, não siga o padrão CORBA, ele possui uma série de características que permitem o desenvolvimento de aplicações distribuídas que utilizem grupos de objetos. Os demais *middlewares* apresentados são o RDO/C++[1] [Isis94], o Orbix+Isis e o Electra. Estes *middlewares* estendem o padrão CORBA para permitir o suporte a grupo no núcleo do ORB. Todos os *middlewares* apresentados são construídos sobre um sistema de comunicação que oferece suporte a grupo. No caso do Orbix+Isis, RDO/C++ e Isis/C++, este sistema é a ferramenta Isis. Estes três *middlewares* se apresentam na forma de produtos comerciais. Já o *middleware* Electra é estruturado de modo que possa ser executado sobre vários sistemas de mais baixo nível para o suporte a grupo.

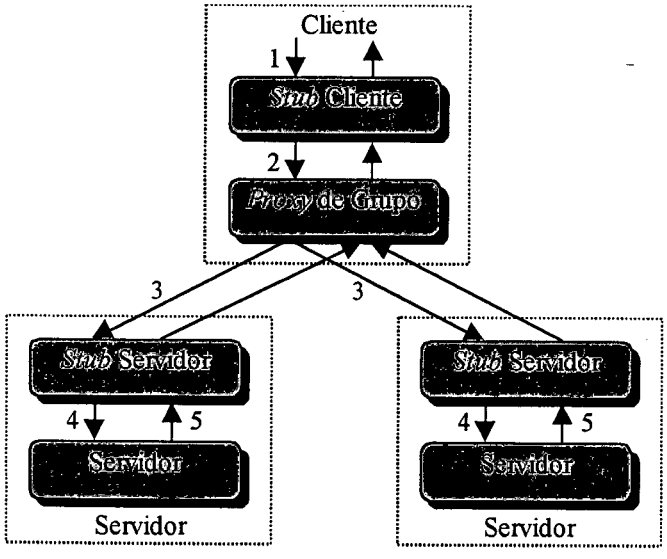
4.2 Isis/C++

O Isis é uma ferramenta para programação distribuída que provê uma série de mecanismos baseados em grupos de processos e difusão confiável para prover tolerância a faltas e alta disponibilidade [Birman91, Hagsand92]. A confiabilidade na ferramenta Isis implica em atomicidade na comunicação de grupo, garantias de ordem de entrega e terminação assíncrona temporizada [Birman91]. O *middleware* Isis/C++ é uma interface que abstrai os serviços da ferramenta Isis para permitir o desenvolvimento de aplicações distribuídas

[1] O nome RDO/C++ vem da expressão em inglês: "*Reliable Distributed Objects in C++*".

orientadas a objeto. Este *middleware* é implementado em C e C++ sobre as versões 2.1 e 3.0 da ferramenta Isis [Hagsand91]. Embora, o Isis/C++ não possa ser considerado um *middleware* CORBA, os seus objetos também possuem uma interface descrita por uma linguagem de definição. Esta linguagem IDL não é compatível com a OMG IDL definida pelo padrão CORBA.

As especificações de interface dos objetos Isis/C++ consistem em uma coleção de entradas, que podem ser usadas por um compilador para gerar *stubs* e *proxies*. Cada entrada corresponde a um método suportado pelo serviço. Os *stubs* são integrados aos objetos clientes e servidores por herança, permitindo a estes objetos interagirem entre si. Os *stubs* servidores são usados para registrar as interfaces dos serviços oferecidos e manipular requisições e respostas. Os *stubs* clientes oferecem uma interface para o acesso a objetos remotos por meio de *proxies*. Os *proxies* consistem em versões locais de objetos remotos com capacidade de comunicação. Em particular, os *proxies* são responsáveis por manipular as interações de grupo, fornecendo transparência de grupo aos clientes [Hagsand91].



Legenda (1) Chamada de método no *stub* cliente
(2) Pedido de transmissão no *proxy* do grupo
(3) Chamada remota no *stub* servidor
(4) Chamada de método no servidor
(5) Envio da resposta do servidor

Figura 4.2 – Invocação de um serviço em um grupo de objetos do Isis/C++

A figura 4.2 ilustra a invocação de um serviço oferecido por um grupo de objetos Isis/C++. Primeiro é feita uma chamada a um método do *stub* cliente, onde é montada uma mensagem com a identificação do serviço e seus argumentos. Então, esta mensagem é

passada ao método de comunicação do *proxy* do grupo, que a repassa aos membros do grupo segundo o protocolo de comunicação utilizado. Em cada sítio receptor, um *stub* servidor retira os argumentos da mensagem e invoca o método adequado implementado no servidor. Finalmente, é montada uma mensagem com os argumentos de resposta que segue o caminho inverso até o objeto cliente que invocou o serviço [Hagsand91].

Os métodos de comunicação fornecidos aos clientes podem ter três variantes: comunicação síncrona do tipo pedido-resposta, comunicação assíncrona do tipo *one-way* e comunicação síncrona bidirecional do tipo pedido-resposta. Estas variantes fazem uso dos protocolos de comunicação de grupo disponíveis na ferramenta Isis.

4.3 RDO/C++

O RDO/C++ é um *middleware* para desenvolvimento de aplicações distribuídas orientadas a objeto. Este *middleware* oferece um suporte de execução que, assim como o Isis/C++, faz uso dos serviços oferecidos pela ferramenta Isis. O RDO/C++ pode ser considerado uma evolução do Isis/C++, onde a linguagem de definição de interface é compatível com a OMG IDL [Isis94]. O *middleware* RDO/C++ contém bibliotecas que simplificam o uso do Isis em muitas aplicações C++ distribuídas. Este *middleware* é capaz de suportar uma variedade de modelos distribuídos, tais como: grupos de réplicas ativas ou máquina de estado, grupos heterogêneos com funções particionadas e ainda grupos na forma de *publisher/subscriber* [Lung96].

A estrutura geral do RDO/C++ é muito similar a outros ORBs. A partir de interfaces IDL, o compilador RDO/C++ gera uma coleção de classes C++. Entre estas classes se encontram os *proxies* e *dispatchers*. Os *proxies* são utilizados pelos clientes, sendo que representam localmente um grupo ou um objeto servidor individual. Em particular, os objetos *proxies* recebem mensagens dos clientes, ordenam os argumentos e os transportam por meio do Isis até os membros do grupo servidor. Os *dispatchers* residem em pontos de entrada de processos Isis e são ligados aos objetos membros de grupo. As mensagens recebidas nestes pontos de entradas são desempacotadas e repassadas para as implementações dos objetos que realizam os serviços indicados. Nas operações que

necessitam de respostas, os *dispatchers* montam mensagens de retorno e as enviam aos clientes [Lea94].

O modelo cliente-servidor implementado no RDO/C++ é similar ao definido no padrão CORBA. Todavia, devido ao RDO/C++ estar fundamentado nos protocolos Isis, é possível aplicá-lo na implementação de sistemas distribuídos que fazem uso de servidores replicados em diferentes modelos de grupo [Lung96]. A figura 4.3 apresenta de forma sucinta a estrutura do RDO/C++. As aplicações utilizam serviços do RDO/C++, e este utiliza as facilidades de comunicação do Isis, localizado acima do sistema operacional.

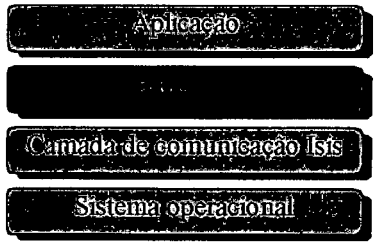


Figura 4.3 – Arquitetura do RDO/C++

4.4 Orbix+Isis

O *middleware* Orbix+Isis é um produto comercial desenvolvido em conjunto pelas empresas *Isis Distributed Systems Inc.* e *IONA Technologies, Ltd.* O Orbix+Isis é um suporte que permite que um sistema seja construído como um conjunto de objetos interagindo segundo o padrão CORBA. Este suporte simplifica o desenvolvimento e a integração de aplicações distribuídas tolerantes a faltas. Cada objeto tem uma interface bem definida e especificada em uma linguagem de definição de interface compatível com a OMG IDL. Assim como o Isis/C++ e o RDO/C++, a implementação do Orbix+Isis é simplificada pelo uso de recursos da ferramenta Isis na implementação das abstrações de grupo. Mecanismos de pertinência, transferência de estado e difusão confiável com diferentes tipos de ordenação, fornecidos pelo Isis, são usados pelo Orbix+Isis [Isis95].

A seguir são discutidos os modelos de suporte a grupo do Orbix+Isis, apresentando suas características e utilizações. Posteriormente, é descrita a arquitetura do Orbix+Isis, detalhando-se alguns de seus componentes.

4.4.1 Modelos de Grupo do Orbix+Isis

No Orbix+Isis, os servidores formam grupos de objetos C++ replicados ou associados onde a transferência das interações entre clientes e servidores é confiável. Os grupos de objetos podem ser definidos sobre dois modelos de execução: réplicas ativas e fluxo de eventos[1]. No modelo de réplicas ativas, cada objeto membro de grupo compartilha a mesma interface e implementação, o que assegura que cada membro responda de maneira idêntica às mesmas requisições de serviço. O Orbix+Isis mantém uma classe base com suporte a réplicas ativas para implementar servidores segundo este modelo de execução. Execuções no modelo de réplicas ativas permitem três tipos de comunicações [Lung96]:

- **Difusão** – A comunicação do tipo difusão corresponde ao modelo de máquina de estado. Neste tipo de comunicação, uma requisição é enviada para todos os membros de um grupo de objetos e um único resultado é devolvido ao cliente que realizou a requisição, mantendo assim, a transparência de cliente. Todavia, o cliente pode ganhar acesso às respostas de todos os membros do grupo, para tal é necessário que construa um *smart proxy*. Os objetos *proxies*, no lado cliente, fornecem suporte para a comunicação de grupo. Neste sentido, o *smart proxy* construído deve herdar o comportamento do *proxy* padrão e adicionar a capacidade de enviar todos os resultados ao cliente.
- **Escolha do Cliente** – Na comunicação do tipo escolha do cliente, as requisições são passadas a apenas um dos membros do grupo. Uma função de escolha do lado cliente determina o membro a receber a requisição. Este tipo de comunicação tem o objetivo de aumentar o desempenho das interações entre clientes e servidores, sendo que foi criado apenas para operações de leitura. Esta restrição faz com que o membro escolhido não precise repassar o resultado das operações realizadas para o resto do grupo.
- **Líder/Seguidores** – Na comunicação do tipo líder/seguidores[2], um dos membros do grupo é considerado líder e os membros restantes são considerados seguidores. Quando uma operação é requisitada, o líder a executa e então envia os resultados ao cliente e a todos os seguidores. Estes seguidores utilizam os resultados para a atualização de seus

[1] O termo fluxo de eventos é referenciado em inglês como "*event stream*".
[2] A expressão líder/seguidores é referenciada em inglês como "*coordinator/cohort*".

estados de modo que possam substituir o líder em caso de falha. Esta abordagem permite o atendimento de tanto operações de leitura como de escrita. Tal como na comunicação com escolha do cliente, na comunicação líder/seguidores uma função de escolha determina o líder para o processamento das requisições. Se o líder atual falhar, a função de escolha é ativada automaticamente para escolher o novo líder.

O modelo de fluxo de eventos segue o estilo de grupo *publisher/subscriber*. Os clientes correspondem aos *publishers* e os servidores correspondem aos *subscribers* ou receptores de eventos. Além destes, existe uma terceira entidade chamada de fluxo de eventos. O fluxo de eventos é uma entidade replicada e tolerante a faltas, implementada a partir de um grupo e executando no modelo de réplicas ativas. Neste contexto, os clientes usando apenas comunicação assíncrona do tipo *one-way*, enviam mensagens ao fluxo de eventos que as mantém e as direciona aos objetos receptores de eventos. Os receptores de eventos podem se juntar ou deixar o grupo a qualquer momento. A participação no grupo pode inclusive ser programada para a recepção de uma quantidade determinada de eventos. Um número específico de mensagens enviadas ao fluxo de eventos é salvo em um histórico para ser repassado aos novos receptores de eventos incluídos no grupo. O modelo de fluxo de eventos desacopla os clientes dos servidores, pois os objetos clientes que mandam mensagens para o fluxo de eventos, não se preocupam se os receptores de eventos estão ou não ativos. [Lung96]. A figura 4.4 ilustra este modelo de execução em grupo:

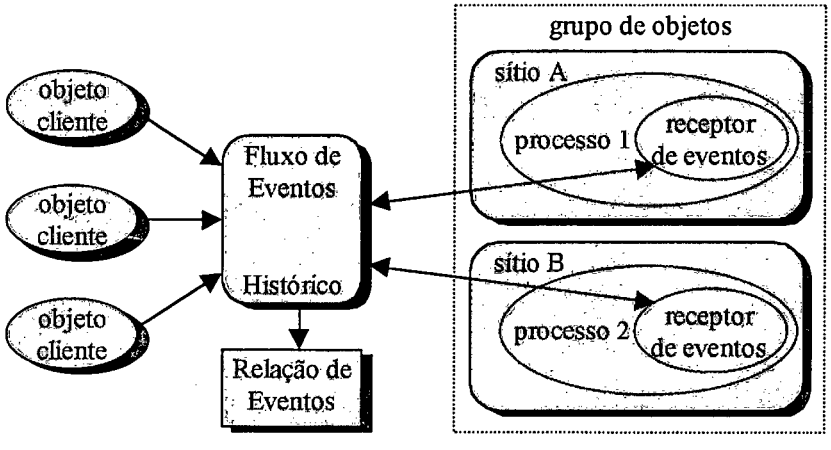


Figura 4.4 – Arquitetura do modelo de fluxo de eventos

4.4.2 Arquitetura do Orbix+Isis

A arquitetura de um sistema que utiliza o Orbix+Isis é ilustrada na figura 4.5. O Orbix+Isis consiste de bibliotecas de classes C++ e de um suporte de execução. Este suporte implementa as funcionalidades dos modelos de execução de grupo: réplicas ativas e fluxo de eventos. Nesta estrutura convivem um ORB convencional, que corresponde a camada de comunicação Orbix [IONA95], e um ORB com suporte a grupo, que corresponde a camada de comunicação Orbix+Isis. A camada Orbix trata de comunicações ponto a ponto, enquanto a camada Orbix+Isis usa facilidades do Isis no suporte a grupo, oferecendo as bases para aplicações distribuídas tolerantes a faltas [Lung96].

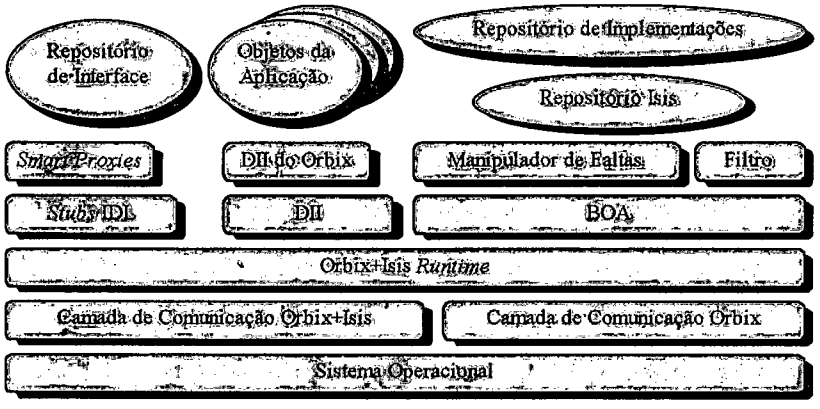


Figura 4.5 – Arquitetura do Orbix+Isis[1]

A linguagem IDL do Orbix, compatível com a OMG IDL, foi estendida para permitir a geração dos códigos necessários para a construção de aplicações tolerantes a faltas no Orbix+Isis. As interfaces IDL de grupo no Orbix+Isis geram na compilação as estruturas de suporte a grupo, como *stubs* e *proxies*. Usando estas estruturas, o cliente conecta-se e comunica-se com o grupo de objetos como se este fosse um objeto Orbix simples [Lung96].

Outro importante elemento da arquitetura do Orbix+Isis é o repositório Isis (IsR). O IsR corresponde a uma hierarquia de arquivos do Orbix+Isis similar ao repositório de implementações do Orbix. Cada servidor Orbix+Isis tem o seu arquivo IsR, que define os procedimentos de ativação e de configuração do grupo. Os aspectos de desempenho e de execução do grupo são especificados no repositório Isis. As informações disponíveis nestes

[1] A sigla DII que aparece no nome de dois componentes da arquitetura do Orbix+Isis vem da expressão em inglês: "*Dynamic Invocation Interfaces*", sendo que referencia interfaces de invocação dinâmica.

arquivos permitem, por exemplo, mudanças na aplicação envolvendo o modelo de execução de grupo, sem a necessidade de modificação do código ou de novas compilações.

4.5 Electra

O Electra é um ORB compatível com o padrão CORBA, cuja a arquitetura suporta grupos de objetos. Para o desenvolvimento de aplicações distribuídas, este modelo combina os benefícios do padrão CORBA com o poder de ferramentas de mais baixo nível, tais como: Isis, Horus, Transis, Consul [Mishra93], Chorus [Chorus92] e Muts [Renesse93b], entre outros. Na atual versão do Electra, é possível torná-lo operacional sobre as ferramentas Isis, Horus e Muts, mas novos adaptadores podem ser desenvolvidos para outras plataformas, o que o torna um sistema flexível [Maffeis95a, Maffeis96].

O Electra é implementado em C++ e o mapeamento IDL para C++ está de acordo com o especificado pela OMG [OMG94]. A IDL do Electra é idêntica às especificações da OMG, pois o modelo considera as implementações de objeto como grupos de objetos. No Electra, os objetos membros de um grupo devem ser do mesmo tipo[1], ou seja, os objetos devem ser implementações da mesma interface IDL ou pelo menos ter uma interface antecessora comum. Neste último caso, só as operações definidas por esta antecessora podem ser difundidas ao grupo [Maffeis95a].

A seguir são discutidos os modelos de suporte a grupo do Electra, apresentando suas características e utilizações. Posteriormente, é descrita a arquitetura do Electra, detalhando-se alguns de seus componentes.

4.5.1 Modelos de Grupo do Electra

O Electra, como todo ORB, é o responsável pela implementação das semânticas de comunicação e independe das linguagens de programação ou das técnicas de implementação usadas na construção dos objetos. Diferente dos ORBs convencionais, o Electra suporta, além de comunicação ponto a ponto, difusão confiável. O cliente faz uso de um mesmo modelo de invocação de método, independente do servidor ser um objeto simples ou um

[1] O tipo de um objeto CORBA é definido por sua interface IDL.

grupo de objetos. Estas invocações podem ser síncronas, assíncronas ou semi-síncronas[1], e serem realizadas em interfaces estáticas ou dinâmicas. Além disso, no Electra a comunicação dos clientes com os grupos de objetos pode se dar de duas formas [Maffeis95a]:

- **Transparente** – A comunicação ocorre de forma transparente quando o cliente não consegue perceber se o servidor é implementado por um objeto simples ou um grupo de objetos. Neste tipo de comunicação, um grupo é visto como um objeto único e altamente disponível. As requisições são submetidas ao grupo como num ORB convencional, ou seja, o cliente recebe apenas um resultado do grupo.
- **Não-transparente** – A comunicação ocorre de forma não-transparente quando o cliente tem acesso aos resultados de cada membro individual do grupo de objetos que atendeu uma invocação.

O Electra suporta grupos de objetos no modelo de réplicas ativas. Neste contexto, o Electra faz uso de ferramentas de mais baixo nível para o fornecimento de facilidades de difusão confiável e gestão de grupo [Lung96]. Esta abordagem é ilustrada na figura 4.6.

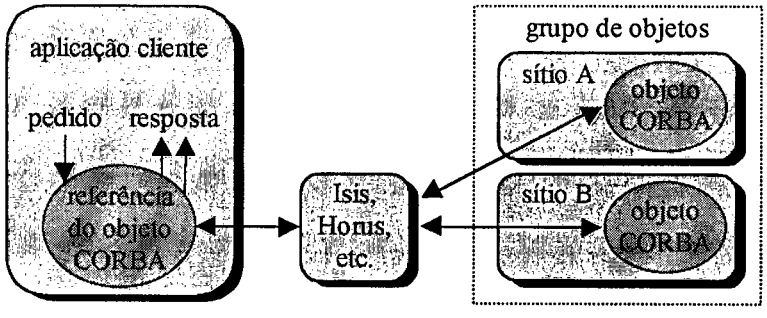


Figura 4.6 – Comunicação de grupo no Electra

Para adicionar suporte de grupo ao padrão CORBA, apenas duas classes da implementação do ORB foram estendidas, as classes *BOA* e *Environment*[2]. Todavia, ambas as classes continuam compatíveis com o padrão CORBA, já que novos métodos foram adicionados sem alterar a estrutura ou a semântica dos métodos padronizados. Operações

[1] As invocações semi-síncronas funcionam como assíncronas por um período determinado de tempo, sendo que depois disto são bloqueadas até o retorno do resultado. Este tipo de invocações é referenciado em inglês como: "*deferred-synchronous invocation*".

[2] A classe *Environment* controla o ambiente de execução do ORB. Quando um cliente invoca uma operação de um objeto remoto, um objeto *Environment* deve ser passado como um dos parâmetros. No CORBA, estes objetos são usados principalmente para passar exceções dos servidores para os clientes.

de gestão de grupo, suportadas por ferramentas como o Isis, são disponíveis na classe *BOA*. Estas operações permitem, entre outros, a ativação e destruição de grupos, a seleção do protocolo de difusão do grupo e a transferência de estado entre os membros do grupo. Operações para a seleção do estilo de invocação de métodos são disponíveis na classe *Environment*. Os três métodos adicionados a esta classe são apresentados a seguir [Maffeis95a]:

- ***call_type*** – A operação *call_type* é utilizada pelo cliente para especificar se a invocação deve ser síncrona, assíncrona ou semi-síncrona.
- ***num_replies*** – A operação *num_replies* especifica a forma da recepção das respostas de um grupo. O argumento *ALL* indica que todas as respostas dos membros operacionais devem ser coletadas. O argumento *MAJORITY* indica que as respostas devem ser coletadas até atingir a maioria. Finalmente, o argumento *COMPARE* indica que todas as respostas dos membros operacionais devem ser coletadas e comparadas, sendo que a resposta mais freqüente deve ser enviada ao cliente.
- ***call_completed*** – A operação *call_completed* é utilizada para verificar se uma invocação semi-síncrona foi completada.

4.5.2 Arquitetura do Electra

O Electra é um sistema flexível, capaz de operar sobre vários tipos de ferramentas de suporte a grupo e sistemas operacionais. A figura 4.7 ilustra a arquitetura de um sistema fazendo o uso do Electra [Lung96].

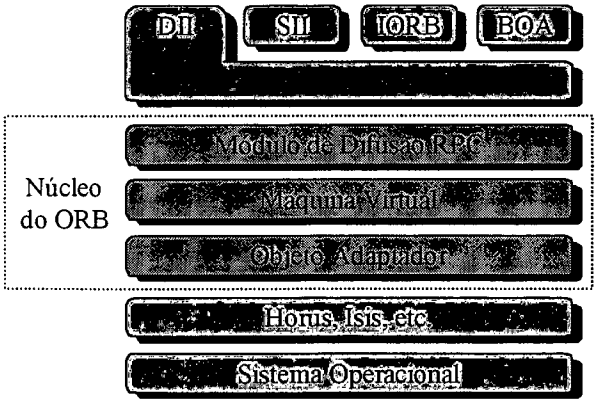


Figura 4.7 – Arquitetura do Electra

No topo da arquitetura do Electra estão as interfaces de invocação estática (SII[1]), as interfaces de invocação dinâmica (DII), a interface ORB (IORB) e o adaptador de objetos básico (BOA). Estes elementos servem de interface para as aplicações e funcionam de acordo com o padrão CORBA. Abaixo destes, se encontra o núcleo do ORB, que consiste em [Lung96]:

- **Módulo de Difusão RPC** - O módulo de difusão RPC suporta RPC assíncrono para grupos de objetos ou objetos simples.
- **Máquina Virtual** - A máquina virtual é inserida nesta arquitetura com o objetivo de oferecer portabilidade do Electra sobre as diversas plataformas básicas possíveis. A sua função é abstrair o ORB de facilidades oferecidas por estas plataformas de mais baixo nível. Dentre estas facilidades estão a comunicação fim-a-fim, grupo de processos, fluxos de execução[2] e outros.
- **Objeto Adaptador** - O objeto adaptador é específico para a plataforma básica utilizada e tem a função de mapear as operações da máquina virtual em operações desta plataforma. Neste contexto, para que o Electra funcione em uma nova plataforma basta desenvolver um outro objeto adaptador e acrescentá-lo a arquitetura. Não há a necessidade de fazer qualquer modificação aos códigos residentes acima do objeto adaptador. As atuais implementações de objeto adaptador para o Isis, Horus e Muts compreendem aproximadamente mil linhas de código C++ cada. Desta forma, pode-se perceber a fácil adaptabilidade destas ferramentas no suporte Electra para aplicações distribuídas.

4.6. Conclusões do Capítulo

A principal vantagem da abordagem de integração é a transparência dos mecanismos de processamento de grupo. Um objeto cliente pode requisitar serviços a um grupo de objetos da mesma forma que faria a um objeto único. Tanto o objeto único quanto o grupo de objetos implementam a mesma interface. O custo da transparência nesta abordagem é que a semântica das referências de objetos CORBA é modificada. A especificação CORBA define

[1] A sigla SII vêm da expressão em inglês: "*Static Invocation Interface*".
[2] Os fluxos de execução são referenciados em inglês como: "*threads*".

uma referência de objeto como um identificador que de forma confiável denota um objeto particular. Na especificação CORBA, a referência de objeto, sempre que usada em uma requisição, identifica o mesmo objeto. Além disso, um objeto pode ser denotado por múltiplas referências distintas [OMG95a]. Todavia, na abordagem de integração uma referência de objeto não identifica somente um objeto CORBA, mas possivelmente um grupo de objetos. Ademais, os objetos referenciados podem mudar com o passar do tempo. A referência de objeto pode até designar um grupo vazio [Felber96].

Os *middlewares* Isis/C++, RDO/C++, Orbix+Isis e Electra tem o objetivo de prover um ambiente para a programação distribuída orientada a objeto, suportando a noção de grupo de forma similar. Esta similaridade é mais acentuada nos casos do Orbix+Isis, RDO/C++ e Isis/C++, já que os seus desenvolvimentos tiveram origem no mesmo grupo de pesquisa na Universidade de Cornell, E.U.A. Todos os *middlewares* estão fundamentados no uso de serviços de mais baixo nível que oferecem algum tipo de facilidade para a gestão e comunicação de grupo. Enquanto os *middlewares* Orbix+Isis, RDO/C++ e Isis/C++ operam apenas sobre a ferramenta Isis, o Electra apresenta uma arquitetura que permite o seu porte para várias ferramentas de suporte a grupo. Atualmente, existem implementações do Electra sobre o Horus, Isis e Muts.

O *middleware* Isis/C++, diferente dos demais, não segue o padrão CORBA, entretanto apresenta várias semelhanças com os ORBs apresentados. Dentre estas semelhanças, está o provimento de uma linguagem de definição de interface. Esta linguagem não é compatível com a OMG IDL. O Electra, o Orbix+Isis e o RDO/C++ estendem o padrão CORBA para permitir o suporte a grupo de objetos no núcleo do ORB. Porém, apenas os dois primeiros oferecem um adaptador de objetos básico para auxiliar nas atividades de gestão de grupo. As linguagens IDL do Electra, Orbix+Isis e RDO/C++ estão de acordo com o padrão CORBA. Todavia no Orbix+Isis e no RDO/C++, as regras de mapeamento para a linguagem de implementação foram estendidas para gerar os códigos de gestão e comunicação de grupo. Em especial, o RDO/C++ é considerado uma evolução do Isis/C++, onde a linguagem IDL é compatível com a OMG IDL. Além disso, apesar do Electra respeitar as regras de mapeamento da linguagem IDL, classes da implementação do ORB foram estendidas para permitir o suporte a grupo.

5. ABORDAGEM DE SERVIÇO

5.1 Introdução

A abordagem de serviço para a adição de suporte a grupo ao ORB está de acordo com a filosofia adotada pela OMG para o acréscimo de funcionalidades ao padrão CORBA. A plataforma básica de comunicação provida pelo CORBA, ou seja, o ORB, suporta apenas os mecanismos para a invocação de métodos de objetos remotos. Funções mais específicas são adicionadas ao ORB na forma de objetos de serviço definidos através de especificações COSS da OMG. Os objetos de serviço servem de blocos de construção para dar suporte ao desenvolvimento de diferentes aplicações. Neste contexto, a abordagem de serviço segue o modelo de outras funcionalidades que foram adicionadas ao CORBA, como os COSS de ciclo de vida, persistência e transações. Estes serviços foram especificados em IDL e adotados como padrões CORBA [OMG95b].

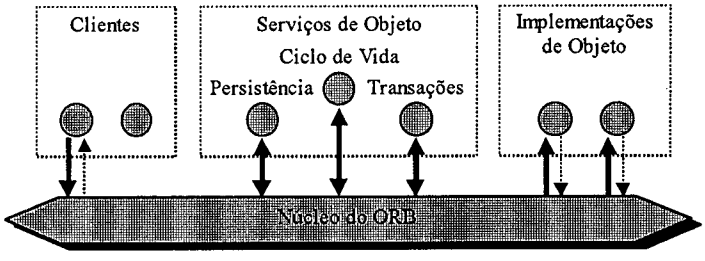


Figura 5.1 – Utilização de serviços CORBA

A idéia básica da abordagem de serviço é prover a comunicação de objetos em grupo como um serviço no topo do ORB, e não como uma parte do próprio ORB. Desta forma, o serviço de suporte a grupo deve ser ligado ao barramento de objetos, ou seja, o núcleo do ORB, aprimorando-o com novas funcionalidades. Na figura 5.1 é ilustrada a utilização de serviços CORBA de ciclo de vida, persistência e transações. Apesar .dos serviços terem sido representados como um único objeto CORBA, na maioria das vezes

estes correspondem a uma coleção objetos e suas interfaces que estão distribuídos ao longo da rede.

A OMG ainda não especificou nenhum serviço ou serviços que suportem alguma forma de processamento replicado ou de tolerância a faltas. O COSS de eventos fornece mecanismos para a comunicação multi-objeto, mas não pode ser utilizado para suportar aplicações confiáveis. Todavia, apesar da indefinição da OMG, vários serviços para a tolerância a faltas em aplicações que seguem a arquitetura OMA, já foram implementados sobre *middlewares* CORBA existentes. Estes serviços são geralmente baseados em mecanismos de comunicação em grupo. Contudo, o serviço *Object Fault-tolerance Service* (OFS) [Liang98], detalhado posteriormente, não oferece suporte a este tipo de comunicação.

Devido a destinar-se a trabalhar em um ambiente distribuído, o projeto de um serviço de suporte a grupo não pode depender de nenhum componente global, crítico ou centralizado. O uso de tal abordagem compromete a confiabilidade da solução e consiste em um gargalo de desempenho. Desta forma, o serviço de suporte a grupo deve ser formado por uma coleção de objetos de serviço, que devem estar localizados em diferentes nodos da rede. Estes objetos devem ser responsáveis pela gestão de grupo e a entrega de mensagens aos objetos membros de grupo, mantendo as propriedades definidas pelo tipo de comunicação de grupo utilizado. Em particular, devem ser providos objetos de serviço especializados ou *proxies* para serem utilizados diretamente na interação com os clientes e servidores. Estes objetos devem agir como representantes locais dos grupos, provendo as primitivas necessárias para a utilização do serviço. Neste contexto, o serviço de suporte a grupo deve consistir em uma estrutura inerentemente distribuída, sem uma localização definida. A execução de uma requisição em um serviço de suporte a grupo é ilustrada na figura 5.2.

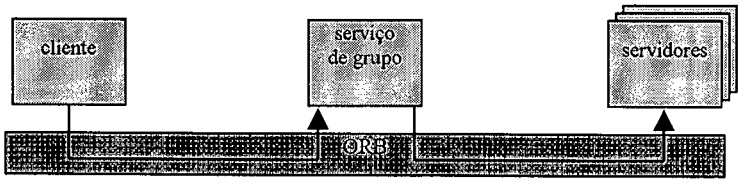


Figura 5.2 – Execução de uma requisição na abordagem de serviço

Alguns objetos de serviço devem ser criados no momento da associação de um cliente ou um servidor ao grupo. Para facilitar tal tarefa, podem ser utilizados objetos fábrica[1]. Tais objetos aprimoram a reusabilidade, já que são independentes da linguagem de programação utilizada para criar os objetos CORBA. Além disso, os objetos fábrica permitem passar argumentos para um objeto em tempo de criação, vencendo as limitações dos construtores em IDL [OMG95b].

A seguir são descritos serviços de tolerância a faltas implementados sobre *middlewares* CORBA, que permitem o desenvolvimento de aplicações confiáveis. Primeiro é apresentado o OFS. Apesar deste serviço prover a tolerância a faltas a partir da replicação de objetos, não são utilizados mecanismos de comunicação de grupo. Posteriormente, é apresentado o serviço *Object Group Service* (OGS) [Felber97]. O OGS consiste em um serviço de suporte a grupo que pode ser executado sobre qualquer ORB compatível com a especificação CORBA 2.0. Finalmente, é feita uma comparação entre os serviços apresentados.

5.2. OFS

A tolerância a faltas normalmente requer algum grau de redundância. Neste contexto, a utilização de múltiplas cópias ou réplicas de uma mesma aplicação é uma abordagem comum. Na literatura [Chen92] é definido que um grupo é formado por objetos pertencentes a uma das três classes: réplicas ativas, semi-ativas e passivas[2]. Diferente das demais, as réplicas passivas não são executadas no sistema. Por sua vez, as réplicas semi-ativas, embora sejam executadas, não respondem a requisições dos clientes. Finalmente, todas as réplicas ativas respondem aos clientes com algum grau de sincronia. Caso ocorram falhas, réplicas passivas podem ser promovidas a semi-ativas e réplicas semi-ativas podem ser promovidas a ativas. Esta abordagem mantém um alto nível de confiabilidade e disponibilidade, todavia pode causar degradação de desempenho.

[1] Os objetos fábrica, referenciados em inglês como "*factory objects*", são definidos pela especificação do COSS de ciclo de vida. Em particular, um objeto fábrica é um objeto CORBA comum que é utilizado para criar outros objetos [OMG95b].

[2] As réplicas ativas, semi-ativas e passivas são designadas na literatura original [Liang98] em inglês como: "*hot backups*", "*warm backups*" e "*cold backups*".

A maioria dos *middlewares* CORBA para a tolerância a faltas utiliza réplicas ativas. Todavia, pesquisas sugerem que réplicas semi-ativas e passivas podem ser utilizadas em várias aplicações [Huang93]. Neste contexto, o OFS é um serviço para tolerância a faltas implementado sobre o Orbix 2.0 e a plataforma Solaris, que suporta réplicas semi-ativas e passivas. A implementação do OFS não requer modificações no ORB ou no mapeamento IDL. Além disso, não é assumido suporte proprietário para a comunicação em grupo.

A seguir é apresentada a arquitetura do OFS, descrevendo-se seus componentes. Posteriormente é detalhado como ocorre o suporte a tolerância a faltas no OFS.

5.2.1. Arquitetura do OFS

Conforme o ilustrado na figura 5.3, o OFS é formado por dois objetos: o gerente de replicação e o anjo guardião[1]. Estes objetos devem estar presentes em cada sítio do sistema. O gerente de replicação é responsável por aceitar pedidos de registro de objetos de aplicação que requerem suporte a tolerância a faltas e manter um número suficiente de réplicas no sistema de acordo com o grau de confiabilidade desejado. Em particular, apenas dois destes gerentes funcionam como réplicas ativas e os demais funcionam como réplicas semi-ativas. O anjo guardião é um objeto interno, isto é, inacessível aos usuários do serviço. As funções deste objeto incluem a monitoração dos objetos de aplicação, a detecção de falhas e a ativação de réplicas. Para tal, o anjo guardião periodicamente realiza comunicações com os objetos de aplicação e os gerentes de replicação ativos.

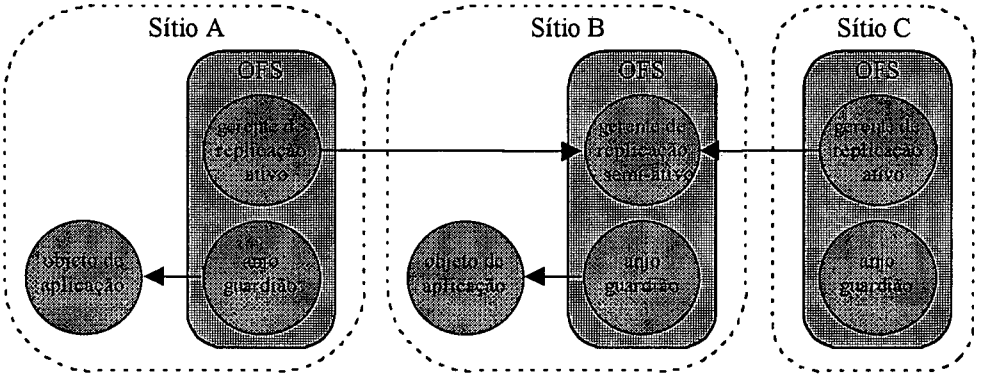


Figura 5.3 – Arquitetura do OFS

[1] Na literatura original [Liang98], o gerente de replicação e o anjo guardião são referenciados em inglês como: "*replication manager*" e "*guardian angel*" respectivamente

Para integrar-se tolerância a faltas à aplicação, os objetos de aplicação devem registrar junto ao OFS uma lista de réplicas e o grau de confiabilidade desejado. Além disso, um objeto de aplicação deve ser capaz de, via OFS, repassar seu estado interno para as réplicas semi-ativas. A interação entre objetos ativos e semi-ativos e o OFS é ilustrada na figura 5.4. As setas indicam que o objeto de origem tem uma referência para o objeto de destino. Quando o OFS é o destino de uma seta, a referência possuída pelo objeto de origem identifica o gerente de replicação.

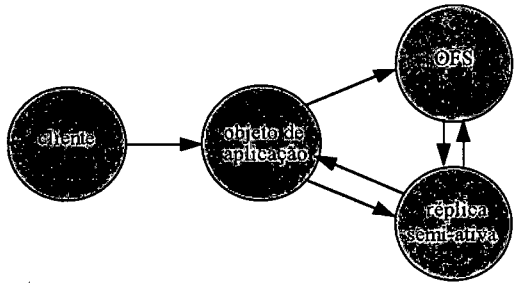


Figura 5.4 – Interações no OFS.

5.2.2. Funcionamento do OFS

Conforme é dito anteriormente, um objeto de aplicação que requer suporte a tolerância a faltas do OFS, deve registrar-se junto ao gerente de replicação, fornecendo uma lista de réplicas e o grau de confiabilidade desejado. Então, o OFS deve ativar um número suficiente de réplicas e continuar a monitorar estes objetos de aplicação. Neste contexto, para um melhor entendimento, a seguir é apresentado um exemplo.

Supondo que um objeto de aplicação `O1` registre os objetos `O2`, `O3` e `O4` como suas réplicas e solicite um grau de tolerância a faltas 1. O OFS deve aleatoriamente escolher uma das réplicas e ativá-la. Desta forma, devem ser mantidas uma réplica semi-ativa e duas réplicas passivas. Neste sentido, o objeto de aplicação é considerado a réplica ativa. Após ser ativada, a réplica semi-ativa pode requisitar o estado corrente a réplica ativa, a fim de sincronizar-se com esta. Além disso, periodicamente, a réplica ativa pode passar para a réplica semi-ativa as requisições que recebeu para serem executadas.

Caso ocorra uma falha em um objeto de aplicação, o OFS deve perceber que existe apenas uma réplica semi-ativa e promovê-la a réplica ativa. Além disso, o OFS deve

escolher aleatoriamente uma das réplicas passivas e promovê-la a semi-ativa. Esta nova réplica semi-ativa deve sincronizar-se com a ativa, requisitando o seu estado. A recuperação de falhas em uma réplica semi-ativa é similar ao apresentado.

5.3. OGS

O OGS é um ambiente orientado a objetos que permite a construção de aplicações distribuídas confiáveis sobre *middlewares* CORBA. O OGS não requer nenhuma mudança ou extensão ao padrão CORBA, podendo ser executado sobre qualquer ORB compatível com a especificação CORBA 2.0. Todavia, este ORB deve oferecer múltiplos fluxos de execução ou *threads*. Esta facilidade é utilizada para implementar comunicações assíncronas. O OGS é construído a partir de vários outros serviços CORBA. Nenhum destes outros serviços é exclusivo para o suporte a grupo, podendo ser utilizados em outros contexto. Desta forma, o OGS consiste em uma solução completamente aberta, já que os mecanismos de grupo são independentes de uma ferramenta de comunicação proprietária.

Figura 5.5 – Serviços utilizados pelo OGS.

A figura 5.5 ilustra os serviços utilizados na implementação do OGS Cada serviço é construído utilizando facilidades providas pelos serviços presentes nas camadas inferiores. O serviço de mensagens permite realizar comunicações assíncronas ponto a ponto. O serviço de difusão provê mecanismos para a difusão de mensagens. O serviço de monitoramento é responsável pela detecção de falhas nos objetos envolvidos em comunicações de grupo. O serviço de consenso garante que mensagens difundidas por vários clientes sejam recebidas segundo um mesmo tipo de ordenação nos vários membros

de grupo. Finalmente, o serviço de grupo de objetos, ou seja, o OGS propriamente dito, oferece mecanismos para a interação entre clientes e grupos de objetos.

A seguir é discutida a arquitetura do OGS. Neste contexto, são apresentados os objetos que compõem o OGS e é descrita em detalhes a interação entre estes objetos. Para tal, são apresentadas situações de execução do OGS. Posteriormente, são discutidos os tipos de comunicação suportados no OGS.

5.3.1. Arquitetura do OGS

O OGS é formado por uma coleção de interfaces e objetos que definem e implementam as operações necessárias para a gestão e comunicação de grupo. Em especial, as interações entre clientes e servidores são intermediadas por objetos *proxies*. A figura 5.6 ilustra a utilização de *proxies* na difusão de uma mensagem. Os *proxies* nos sítios clientes e servidores têm funções distintas. Um *proxy* cliente além de fornecer mecanismos para a difusão, também oferece meios para que um objeto cliente possa obter informações sobre a composição do grupo com o qual está se comunicando. Com tais informações, o cliente pode enviar mensagens ponto a ponto para determinados membros do grupo utilizando os mecanismos de invocação normais do CORBA. Esta facilidade pode ser útil para algumas aplicações.

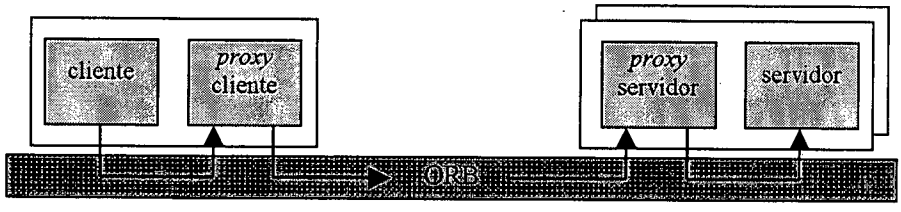


Figura 5.6 – Difusão de mensagem no OGS

As funcionalidades dos *proxies* servidores também não estão restritas a apenas a entrega de mensagens de difusão aos membros de grupo. Para juntarem-se ou deixarem um grupo, os objetos servidores invocam operações em um *proxy* servidor. Estas operações provocam mudanças da pertinência do grupo que são notificadas pelos *proxies* servidores a todos os membros do grupo. Além disso, quando um novo membro junta-se ao grupo, os *proxies* servidores iniciam o protocolo de transferência de estado. Para tal, os *proxies* servidores invocam operações de obtenção e atualização de estado. Neste contexto, todos

os membros de grupo devem ser derivados de uma mesma interface que define as operações invocadas pelos *proxies* servidores.

Um *proxy* permite o acesso a um único grupo. Ou seja, um servidor necessita de um *proxy* para cada grupo de qual deseja fazer parte. Similarmente, um cliente precisa de um *proxy* distinto para cada grupo com o qual deseja comunicar-se. Neste contexto, a quantidade e a natureza dos *proxies* ativos em um dado momento no sistema, depende das aplicações sendo correntemente executadas. Desta forma, a princípio os *proxies* clientes e servidores não estão disponíveis para o uso. Estes *proxies* são criados a medida que são necessários. A criação de *proxies* é realizada por meio de objetos fábrica. Os objetos fábrica estão disponíveis na inicialização do sistema e podem criar *proxies* clientes e servidores para qualquer grupo.

As figuras 5.6 e 5.7 ilustram interações entre os objetos envolvidos no OGS. Uma caixa representa uma operação ativa no objeto cujo o tipo é indicado no topo de cada linha vertical. O tempo flui de cima para baixo. As setas planas representam invocações, as setas em arco representam a criação de novos objetos e as setas com sentido duplo representam mensagens específicas do protocolo de serviço.

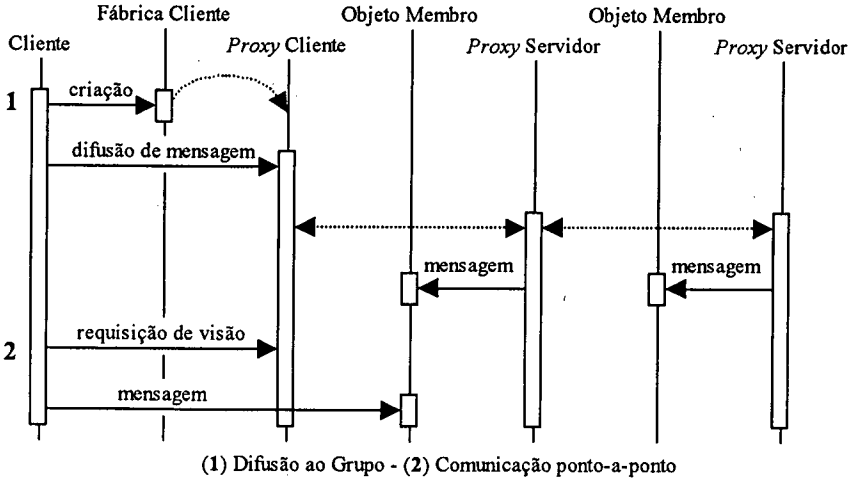


Figura 5.7 – Interação entre um cliente e objetos membros de grupo

Na figura 5.7 é exposta a interação entre um cliente e um grupo. São apresentados dois tipos de interação: uma mensagem de difusão para o grupo e uma comunicação ponto a ponto com um dos membros. Assume-se que o grupo, do exemplo da figura, contenha dois membros. Para executar a difusão, o cliente primeiro requisita a criação de um *proxy*

cliente. Então, a difusão é executada utilizando-se o objeto *proxy*. Para executar a comunicação ponto a ponto, o cliente adquire junto a seu *proxy* referências para os objetos membros de grupo. Então, o cliente envia uma mensagem a um dos membros, utilizando mecanismos de comunicação normais do CORBA.

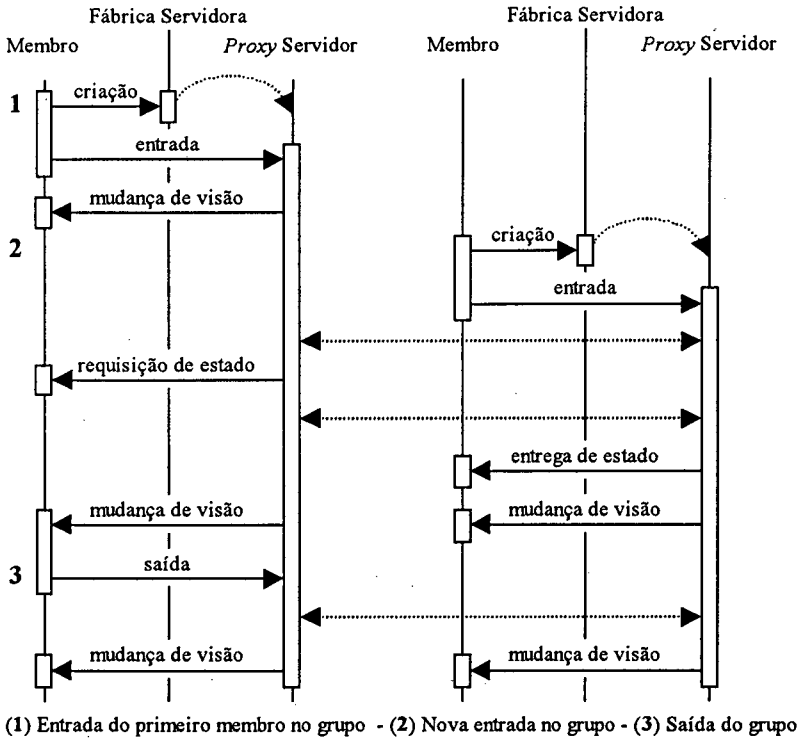


Figura 5.8 – Interação entre dois objetos membros de grupo

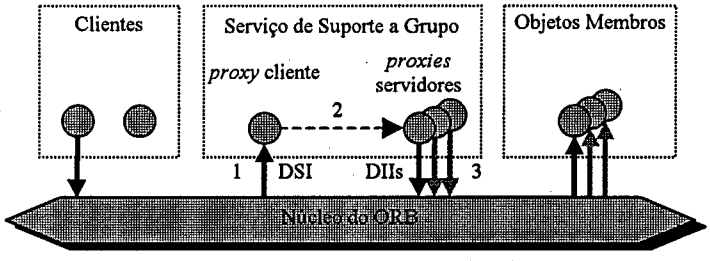
A figura 5.8 mostra como objetos servidores juntam-se e deixam um grupo, e como as mudanças de visão e transferências de estado ocorrem. O primeiro servidor cria um objeto *proxy* e junta-se a um grupo vazio, ocasionando uma mudança de visão. O segundo objeto também junta-se ao grupo, mas como o grupo não está mais vazio, o protocolo de transferência de estado é ativado antes da nova visão de grupo ser enviada aos membros. Finalmente, o primeiro objeto deixa o grupo, ocasionando uma outra mudança de visão. Caso este objeto tivesse falhado, o mesmo protocolo de mudança de visão seria executado.

5.3.2. Tipos de Comunicação no OGS

O OGS permite que a interação entre clientes e membros de grupo ocorra por meio de comunicação tipada ou não-tipada. A comunicação não-tipada permite que os clientes

enviem apenas valores do tipo `any`[1] como mensagens. Neste contexto, para requisitar um serviço de um grupo de objetos, um cliente deve invocar a operação `multicast()` em um *proxy* cliente passando uma mensagem como parâmetro. Esta mensagem é então enviada aos *proxies* servidores que a repassam aos membros de grupo por meio da operação `deliver()`. Finalmente, os membros de grupo executam o serviço adequado, de acordo com conteúdo da mensagem.

Enquanto a comunicação não-tipada é útil e mais eficiente em algumas situações específicas, geralmente é mais conveniente para os clientes invocarem diretamente uma operação da interface do servidor, isto é, utilizar comunicação tipada. A comunicação tipada é um aspecto importante do OGS, já que provê transparência de grupo para os clientes e servidores. Neste contexto, o cliente pode realizar invocações a um grupo de objetos como se estivesse invocando um servidor único. Além disso, os objetos servidores também não podem diferenciar as comunicações de grupo de mensagens comuns.



(1) Requisição do cliente via esqueleto dinâmico
(2) Chamada de difusão aos sítios servidores
(3) Invocação das operações via interfaces dinâmicas

Figura 5.9 – Implementação da comunicação tipada[2]

A comunicação tipada é obtida por meio da utilização de duas características do padrão CORBA: os esqueletos IDL dinâmicos e as interfaces de invocação dinâmica. Os esqueletos dinâmicos permitem que os *proxies* clientes recebam requisições de clientes baseadas na interface IDL do servidor, mesmo que esta interface não seja conhecida em tempo de compilação. Então, estes *proxies* traduzem o conteúdo das requisições para um formato predefinido e realizam uma difusão para os sítios servidores. Nestes sítios, os *proxies* servidores invocam as operações adequadas nos objetos membros por meio de

[1] O tipo *any*, definido pela especificação CORBA, pode suportar valores de quaisquer outros tipos [OMG95a].
[2] A sigla DSI vêm da expressão em inglês: "*Dynamic Squeleton Interface*", sendo que referencia os esqueletos dinâmicos.

interfaces de invocação dinâmica. Qualquer resultado da operação é retornado para o cliente utilizando comunicação ponto a ponto. O funcionamento da comunicação tipada é ilustrado na figura 5.9. Embora nesta figura os sítios clientes e servidores não tenham sido identificados, é importante notar que os objetos clientes e membros de grupo podem estar sendo executados em processadores diferentes e até mesmo ORBs distintos.

5.4. Conclusões do Capítulo

A abordagem de serviço é compatível com o padrão CORBA, já que segue a filosofia da OMG, ou seja, o ORB deve prover os mecanismos mínimos para quebrar as limitações das linguagens e espaços de endereçamento, permitindo que objetos heterogêneos interoperem. Todas as funcionalidades adicionais devem ser providas como serviços, isto é, objetos CORBA específicos com suas interfaces IDL. Por exemplo, a concorrência e a persistência são necessárias para várias aplicações, mas a OMG não inclui suporte para elas no núcleo do ORB. Ao contrário disto, a OMG define COSS que tratam destes temas, mantendo o núcleo do ORB o mais simples possível com apenas as funcionalidades comuns a todas as aplicações [OMG95b]. Ademais, múltiplos serviços podem cooperar no sistema. Por exemplo, o serviço de suporte a grupo pode utilizar o serviço de recuperação para restabelecer objetos servidores após uma falha, ou o serviço de persistência para controlar réplicas persistentes [Felber96]. Além de estar de acordo com a norma CORBA, um serviço é muito mais fácil de ser padronizado do que uma extensão ao ORB ou até mesmo um novo adaptador de objetos.

Os ambientes OFS e OGS consistem em serviços CORBA que permitem o desenvolvimento de aplicações confiáveis sobre *middlewares* CORBA. O OFS oferece suporte a tolerância a faltas por meio de mecanismos de replicação. Todavia, não é utilizada comunicação em grupo. O OGS oferece suporte a aplicações confiáveis por meio de um serviço de grupo de objetos. Este serviço segue a filosofia da OMG, sendo implementado a partir de vários outros serviços CORBA que podem ser utilizados em outros contextos. Tanto o OFS como o OGS estão de acordo com o padrão CORBA, consistindo em soluções abertas. Neste contexto, para a sua implementação, não são utilizadas ferramentas de suporte a grupo proprietárias e as regras de mapeamento IDL são respeitadas.

6. ABORDAGEM DE INTERCEPTAÇÃO

6.1 Introdução

A abordagem de interceptação consiste no provimento de suporte a grupo em *middlewares* CORBA por meio da utilização de mecanismos de reflexão computacional. Em particular, é utilizado o modelo de reflexão comportamental, ou seja, os aspectos estruturais dos programas não são afetados.

A reflexão computacional, ou simplesmente reflexão, consiste na capacidade de um sistema analisar e agir sobre si mesmo, ajustando-se a condições variáveis de seu ambiente [Maes87a, Smith82]. A programação reflexiva, segundo o paradigma de meta-objetos, permite separar o código funcional do não funcional nas aplicações [Maes87b]. O código funcional, ou nível base, relaciona-se com as computações sobre o domínio da aplicação. O código não funcional, ou nível meta, é responsável por supervisionar a execução do código funcional. Para permitir esta supervisão, alguns aspectos das computações do nível base devem ser reificados. A reificação[1] é o processo de tornar explícito algo que normalmente não é parte da linguagem ou do modelo de programação. Existem dois tipos de reflexão computacional [Ferber89]. A reflexão estrutural relaciona-se com a reificação de aspectos estruturais do programa, como herança e tipos de dados. A reflexão comportamental relaciona-se com a reificação das computações e seu comportamento. O aumento da reusabilidade, modularidade, clareza e qualidade do código são algumas das principais vantagens da reflexão computacional.

Várias técnicas de tolerância a faltas tem sido implementadas refletidas, isto é, separadas dos aspectos funcionais de suas aplicações. Em [Fabre95], são apresentadas

[1] Os termos reificação e reificar tem sido utilizados correntemente em português para designar as palavras inglesas "*reification*" e "*reify*" [Lisbôa97].

discussões sobre o uso da reflexão computacional para implementar modelos de replicação em sistemas distribuídos. Em [Fraga97] é estendida esta experiência para ambientes CORBA, porém envolvendo mecanismos não suportados pelo padrão CORBA.

A abordagem de interceptação prevê que as mensagens enviadas aos objetos CORBA devem ser capturadas e mapeadas em um sistema de comunicação de grupo, de maneira transparente para a aplicação. Para tal, podem ser utilizados estruturas das linguagens de programação, interfaces do sistema operacional ou mecanismos do próprio ORB, conhecidos como interceptores. O funcionamento desta abordagem é ilustrado na figura 6.1.

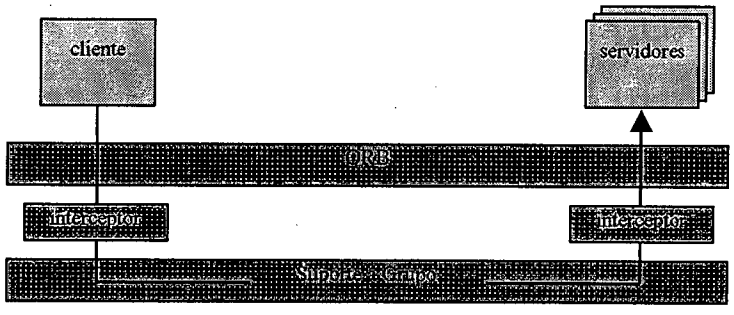


Figura 6.1 – Execução de uma requisição na abordagem de interceptação

O conceito de interceptores foi introduzido inicialmente nas especificações do serviço de segurança do CORBA [OMG95b]. Atualmente um documento *Request for Proposal* foi emitido pela OMG com o objetivo da generalização deste mecanismo [OMG98a]. Logicamente, um interceptor é interposto no caminho de invocação ou reposta entre um cliente e um objeto alvo, sendo responsável pela ativação transparente de controles ou processamentos especiais a quais estariam sujeitas invocações normais no ambiente CORBA.

Quando vários serviços CORBA são requeridos, vários interceptores podem ser utilizados. Em particular, existem dois tipos de interceptores disponíveis. Os interceptores de nível de requisição[1] realizam transformações em uma requisição estruturada. Já os interceptores de nível de mensagem[2] realizam transformações em mensagens derivadas de requisições ou respostas. Estes interceptores são criados pelo ORB quando necessário. Na

[1] Os interceptores de nível de requisição são referenciados em inglês como "*request-level interceptors*".
[2] Os interceptores de nível de mensagem são referenciados em inglês como "*message-level interceptors*".

figura 6.2 são mostrados interceptores sendo chamados durante os caminhos de invocação e resposta.

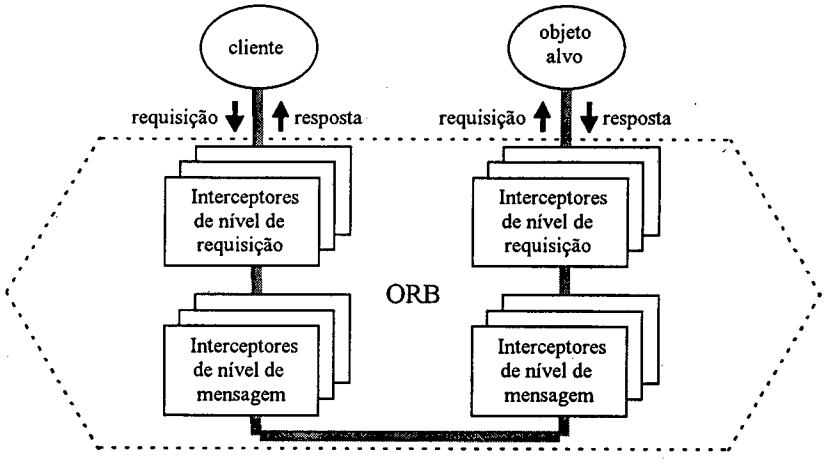


Figura 6.2 – Chamada de interceptores

Um ORB que utilize interceptores precisa conhecer quais interceptores precisam ser chamados e em que ordem. Todavia, quando o cliente executa uma requisição, os interceptores de nível de requisição do lado cliente são sempre chamados antes dos interceptores de nível de mensagem. Já no lado do alvo, os interceptores de nível de mensagem são chamados primeiro. Quando está servindo a um cliente, o ORB usa informações na referência do objeto alvo, bem como políticas locais, para decidir que interceptor precisa ser chamado durante o processamento de uma requisição particular enviada a um objeto alvo específico.

A seguir são descritos ambientes que suportam facilidades para a tolerância a faltas sobre ORBs CORBA, ativadas a partir de mecanismos de interceptação. Primeiro é apresentado o ambiente Eternal. Este ambiente utiliza mecanismos do sistema operacional UNIX para interceptar as chamadas de sistema dirigidas aos objetos CORBA e mapeá-las em uma ferramenta de comunicação em grupo. Posteriormente, é apresentado o Phoinix [Liang96]. O Phoinix define três níveis de serviço de tolerância a faltas, sendo que apenas o último nível prevê o suporte a comunicação em grupo. A interceptação no Phoinix é feita por *proxies* confiáveis e esqueletos tolerantes a faltas gerados a partir de um compilador IDL proprietário. Finalmente, é feita uma comparação entre os ambientes apresentados.

6.2. Eternal

O ambiente Eternal aprimora qualquer implementação da especificação CORBA 2.0, provendo tolerância a faltas às aplicações sem que seja necessária nenhuma modificação no ORB. Para prover a tolerância a faltas, o Eternal replica objetos em diferentes sítios do sistema distribuído e utiliza facilidades providas pela ferramenta Totem. Esta ferramenta consiste em um sistema para comunicação de grupo que provê mecanismos de difusão confiável de mensagens com ordenação total. A vantagem do Eternal é que as operações de grupo são transparentes tanto para os objetos CORBA como para o próprio ORB. Esta funcionalidade é implementada inteiramente em nível de processos de aplicação. Os objetos de aplicação e o ORB também não precisam ser recompilados para tirar vantagem da capacidade de interceptação. Todavia, a implementação do Eternal depende de mecanismos fornecidos pelo sistema Unix.

O Eternal provê dois esquemas para a replicação de objetos: replicação ativa e passiva. Na replicação ativa, as operações são executadas por cada réplica dos objetos, requerendo a detecção e supressão de operações duplicadas. Na replicação passiva, apenas a réplica primária executa cada operação, requerendo mecanismos adicionais para garantir a consistência entre as réplicas primária e passivas. Os objetos replicados no Eternal podem ser construídos hierarquicamente, isto é, serem compostos por outros objetos. Neste contexto, são providos mecanismos para manipular operações aninhadas. Além disso, é permitida a replicação de objetos clientes e servidores. Estas abordagens permitem o desenvolvimento de aplicações distribuídas como se elas fossem ser executadas em um único processador, mantendo a transparência da replicação.

A utilização de objetos replicados, além de prover tolerância a faltas, permite ao Eternal mascarar falhas em objetos ou processadores. Esta facilidade pode ser utilizada para permitir a remoção e substituição de um objeto ou processador, sem que seja necessário interromper a execução do sistema. No futuro, espera-se ainda que quaisquer componentes de *hardware* e *software* do sistema possam ser substituídos e atualizados. Neste contexto, busca-se um sistema que possa ser executado para sempre, isto é, um sistema eterno.

A seguir é apresentada a estrutura do Eternal, discutindo-se o seu funcionamento. Posteriormente, a interceptação no Eternal é descrita em mais detalhes.

6.2.1. Arquitetura do Eternal

A especificação CORBA 2.0 prevê a interoperabilidade entre objetos ligados a diferentes ORBs. Para tal, os ORBs devem suportar os protocolos GIOP[1] e IIOP[2]. O GIOP é um protocolo que consiste em uma coleção de especificações gerais que permitem que as mensagens do ORB sejam mapeadas em um meio orientado a conexão. Este meio deve apresentar garantias mínimas como: confiabilidade, orientação a cadeias de *bytes* e notificação de perda de conexão. O protocolo IIOP define as regras para o mapeamento do GIOP sobre os protocolos TCP/IP. A utilização do IIOP permite a ORBs utilizarem a Internet como canal para suas comunicações.

O ambiente Eternal consiste em um sistema de suporte a grupo localizado entre o ORB e o sistema operacional. Este ambiente pode capturar transparentemente uma coleção de chamadas de sistema feitas usando o ORB durante a execução do objeto. O formato e a semântica das chamadas interceptadas são definidos pelo protocolo IIOP. Depois de capturadas, as chamadas de sistema são manipuladas e mapeadas no Totem.

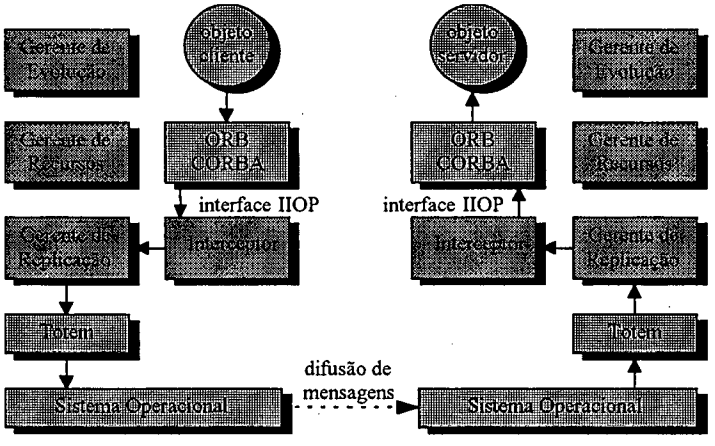


Figura 6.3 – Arquitetura do Eternal

A estrutura do Eternal é ilustrada na figura 6.3 [Narasimham97b]. Os clientes realizam chamadas via ORB, que utiliza o protocolo IIOP. O interceptor captura as chamadas IIOP, que são originalmente dirigidas ao TCP/IP, e as repassa ao gerente de replicação. Então, este gerente é responsável por difundir as mensagens por meio do

[1] A sigla GIOP vem da expressão em inglês: "*Generic Inter-ORB Protocol*".
[2] A sigla IIOP vem da expressão em inglês: "*Internet Inter-ORB Protocol*".

Totem. No lado servidor, cada réplica é invocada e os resultados são emitidos ao cliente pelo caminho inverso.

O gerente de recursos do Eternal, ilustrado na figura 6.3, cria as réplicas de um objeto e as distribui ao longo da rede de acordo com o nível de replicação desejado. O gerente de replicação distribui as operações para as réplicas, mantém a consistência entre estas, detecta e recupera falhas e sustenta a operação de todos os componentes quando ocorre particionamento de rede. Finalmente, o gerente de evolução explora a replicação para suportar atualizações de componentes de *hardware* e *software* no sistema.

6.2.2. Interceptação no Eternal

Um objeto CORBA típico invoca muitas chamadas de sistema durante seu tempo de vida. Isto inclui chamadas para requisição de memória, acesso a bibliotecas, operações de arquivo e operações de rede. Todavia, apenas uma pequena coleção destas chamadas necessita ser interceptada e manipulada pelo Eternal. As chamadas capturadas devem envolver a comunicação com outros objetos, que podem estar presentes remota ou localmente. Tais chamadas são invocadas pelo ORB, segundo o protocolo IIOP, para estabelecer conexões entre objetos e manter a interação entre estes.

open (descritor)
close (descrityor)
read (descritor, <buffer de leitura>)
write (descritor, <buffer de escrita>)
poll (<lista de descritores>)

Figura 6.4 – Chamadas de sistema interceptadas pelo Eternal[1]

Na figura 6.4 são ilustradas as chamadas de sistema interceptadas pelo Eternal. Além de serem utilizadas pelo protocolo IIOP, estas chamadas também são invocadas para realizar operações ordinárias do sistema operacional, devendo, neste caso, serem desconsideradas pelo interceptor. A chamada *open()* é utilizada pelo protocolo IIOP para estabelecer conexões TCP/IP. A chamada *poll()* é utilizada para associar dois objetos em uma conexão. As chamadas *read()* e *write()* são utilizadas na recepção e envio de dados.

[1] Somente os argumentos relevantes das chamadas de sistema são mostrados.

Finalmente, a chamada *close()* é utilizada para encerrar uma conexão. Todas estas chamadas recebem como parâmetro um descritor de conexão[1]. Este descritor é retornado pela chamada *open()*, sendo também utilizado para identificar quais chamadas de sistema são invocadas pelo protocolo IIOP.

6.3. Phoinix

O Phoinix é um ambiente de desenvolvimento de aplicações distribuídas baseado em CORBA, no qual as implementações de objetos são construídas tolerantes a faltas de forma semi-automática. Em particular, são definidos três níveis de serviços de tolerância a faltas no Phoinix, os serviços de reinicialização, recuperação por retrocesso e replicação[2], sendo que a capacidade de tolerância a faltas cresce a medida que o nível aumenta. Atualmente, apenas os dois primeiros níveis de tolerância a faltas do Phoinix estão implementados, isto é, os serviços de reinicialização e recuperação por retrocesso. A versão corrente do Phoinix foi desenvolvida sobre a ferramenta Orbix 3.0 [IONA95] e o sistema operacional SunOS 4.2. Neste contexto, o Phoinix pode ser facilmente portado para outras plataformas, já que o Orbix consiste em uma implementação completa do CORBA.

Um objeto é chamado tolerante a faltas se ele é construído com algum nível de tolerância a faltas. As implementações de objeto dos primeiro, segundo e terceiro níveis de tolerância a faltas do Phoinix são chamadas respectivamente de objetos reinicializáveis, registráveis e replicados[3]. Os objetos reinicializáveis, como o seu nome sugere, continuam a sua execução como um novo servidor depois da recuperação de um erro. Desta forma, contextos de execução anteriores ao erro são perdidos. Já os objetos registráveis continuam o seu serviço, após a recuperação de um erro, a partir do último ponto de recuperação atingido. No caso dos objetos replicados, várias réplicas de uma mesma implementação de

[1] No sistema *Unix*, o descritor de uma conexão tem o mesmo formato e pode receber o mesmo tratamento de um descritor de arquivo.
[2] Os serviços de reinicialização, recuperação por retrocesso e replicação são especificados na literatura original [Liang96] em inglês como "*restart service*", "*rollback-recovery service*" e "*replication service*" respectivamente.
[3] As implementações de objeto reinicializáveis, registráveis e replicadas são especificadas na literatura original [Liang96] em inglês como "*restart objects*", "*logable objects*" e "*replicated objects*" respectivamente.

objeto coexistem no sistema com alguma forma de cooperação, mantendo o serviço mesmo na presença de réplicas faltosas.

O Phoinix foi projetado assumindo-se algumas hipóteses em seu ambiente de operação. Primeiramente, os sítios falhos devem operar segundo um modelo de falha controlada[1] [Schlichting83]. Neste contexto, as falhas no sistema distribuído são relativamente infreqüentes e independentes uma das outras. Além disso, é assumido que as invocações dos cliente às implementações de objetos não são aninhadas. Isto é, uma implementação de objeto invocada não pode requisitar serviços de outras implementações de objetos. Esta hipótese garante que uma implementação de objeto não possa ser suspensa devido a falhas do cliente. Em outras palavras, a detecção de falhas no cliente se torna desnecessária.

Os processos de detecção e recuperação de faltas no Phoinix são realizadas por códigos de tolerância a faltas gerados a partir de um compilador IDL proprietário. No lado cliente, *proxies* confiáveis interceptam e analisam exceções geradas pelo ORB, ativando os mecanismos de recuperação de erros quando necessário. No lado servidor, esqueletos tolerantes a faltas registram as invocações recebidas e o estado corrente das implementações de objeto. Neste contexto, a interceptação é realizada de forma transparente tanto para os objetos clientes como servidores.

A seguir é detalhada a arquitetura do Phoinix e discutido o desenvolvimento de aplicações tolerantes a faltas neste ambiente. Posteriormente, é apresentada a detecção e recuperação de erros por meio dos serviços de reinicialização e recuperação por retrocesso do Phoinix.

6.3.1. Arquitetura do Phoinix

O desenvolvimento de uma aplicação tolerante a faltas no Phoinix é ilustrado na figura 6.5. Como pode ser observado, este modelo é semelhante ao desenvolvimento de uma aplicação convencional em um *middleware* CORBA. Para declarar um objeto tolerante a faltas, as palavras chaves `Restart` ou `Logable` devem ser incluídas na especificação IDL de

[1] O modelo de falha controlada usado é o "*fail-stop model*" [Schlichting83].

objetos reinicializáveis ou registráveis, respectivamente. Então, o compilador EIDL[1] analisa o arquivo de especificações de interfaces e produz códigos de tolerância a faltas em adição aos *stubs* clientes e esqueletos de implementação produzidos pelos compiladores IDL padrões. Para o lado cliente, o compilador EIDL produz um *proxy* confiável para cada interface IDL. Para o lado das implementações de objeto, o compilador EIDL gera esqueletos tolerantes a faltas. A biblioteca de tolerância a faltas define as classes bases dos esqueletos tolerantes a faltas e objetos registráveis.

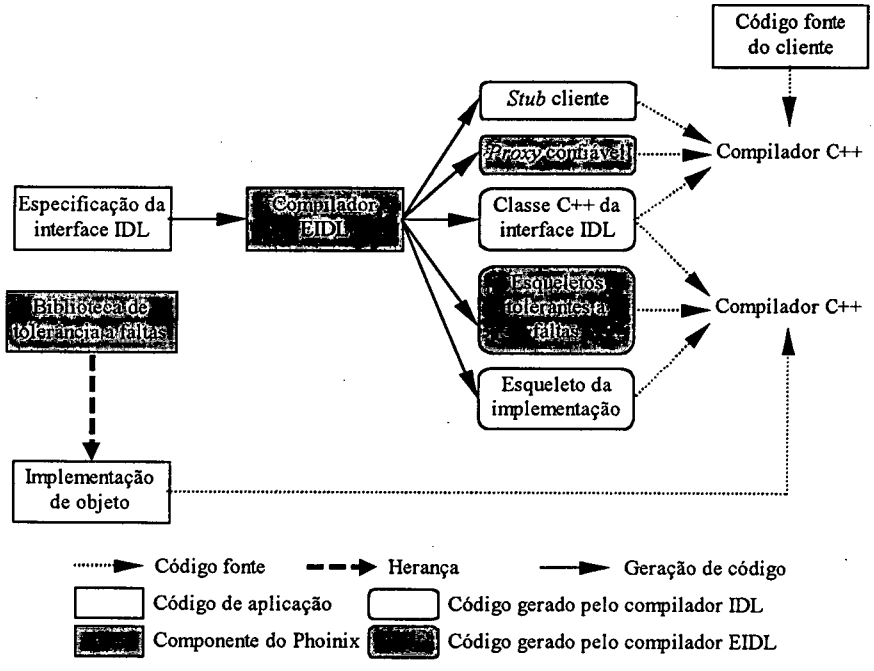


Figura 6.5 – Desenvolvimento de aplicações no Phoinix

A figura 6.6 ilustra a arquitetura de um sistema que faz uso do Phoinix. No topo desta arquitetura encontram-se os objetos clientes e as implementações de objetos. Abaixo destes objetos encontra-se uma camada de tolerância a faltas, que consiste na ferramenta Phoinix propriamente dita. Além dos *proxies* confiáveis, esqueletos tolerantes a faltas e a biblioteca de tolerância a faltas apresentados anteriormente, estão presentes nesta camada um servidor de registros[2] e um gerente de replicação. O servidor de registros trabalha em conjunto com os esqueletos tolerantes a faltas para permitir a correta operação dos objetos

[1] A sigla EIDL vem da expressão em inglês "*enhanced IDL*" e refere-se a uma especialização da linguagem IDL que permite definir objetos tolerantes a faltas [Liang96].
[2] O servidor de registro é referenciado em inglês na literatura original [Liang96] como "*log server*".

registráveis. O gerente de replicação, ainda não está presente na versão atual do Phoinix, todavia deve ser responsável por coordenar as decisões entre as réplicas de uma implementação de objeto durante as invocações de requisições dos clientes. Finalmente, a última camada da arquitetura Phoinix consiste no ORB.

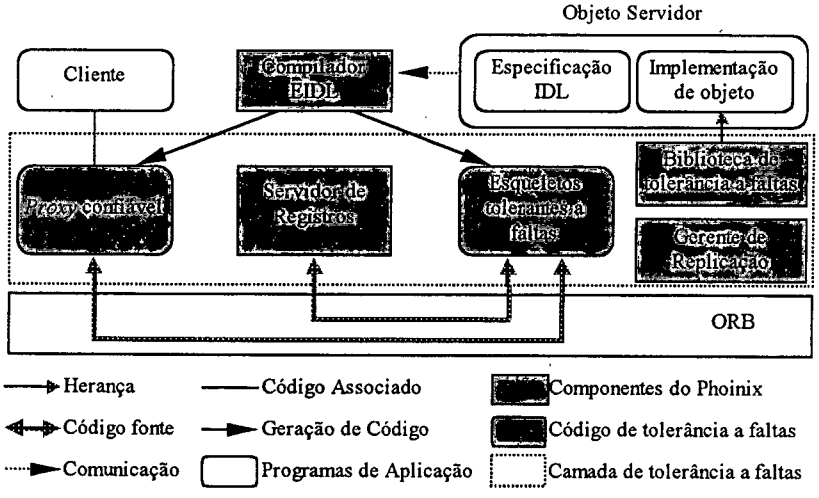


Figura 6.6 – Arquitetura do Phoinix

6.3.2. Detecção e Recuperação de Falhas no Phoinix

A detecção de falhas nas implementações de objeto no Phoinix é feita no lado cliente pelos *proxies* confiáveis. Para tal, estes *proxies* utilizam os serviços convencionais do ORB. Neste contexto, o cliente detecta uma falha em uma implementação de objeto depois de fazer uma invocação e receber uma exceção do ORB. Assim que detectam uma falha, os *proxies* confiáveis também são responsáveis pela ativação do mecanismo de recuperação de erros.

O mecanismo de recuperação de erros varia de acordo com o nível do serviço de tolerância a faltas suportado. Em especial, a recuperação de erros na abordagem de reinicialização é extremamente simples. Assim que é detectada uma falha em uma implementação de objeto reinicializável, o *proxy* cliente apenas ativa uma implementação de objeto substituta. Neste contexto, o estado desta implementação de objeto provavelmente será diferente do estado do objeto substituído antes da ocorrência do erro. Desta forma, este nível de tolerância a faltas apenas garante que as implementações de objetos estarão sempre disponíveis, não preocupando-se com o estado dos objetos.

A recuperação de erros no serviço de recuperação por retrocesso envolve quatro operações: o registro das requisições recebidas pelas implementações de objeto registráveis, a re-execução destas requisições, o armazenamento dos estados dos objetos registráveis e carregamento destes estados. A seguir, é descrito como estas operações são realizadas.

Durante a operação normal do sistema, isto é, na ausência de erros, as requisições dos clientes enviadas às implementações de objetos registráveis são repassadas pelos esqueletos tolerantes a faltas ao servidor de registro, que é responsável por armazená-las. Os objetos registráveis tem a capacidade de decidir se uma requisição deve ou não ser salva de acordo com o domínio de aplicação. Neste contexto, nem todas as requisições são salvas. Além disso, periodicamente o servidor de registro cria pontos de recuperação, isto é, salva o estado dos objetos registráveis. Uma vez que um ponto de recuperação é criado, o servidor de registros elimina as requisições recebidas até então pelo objeto registrável.

Caso uma implementação de objeto falhe definitivamente, caracterizando um *crash*, o cliente recebe uma exceção como resposta a próxima requisição invocada. Então o *proxy* confiável associado ao cliente ativa uma implementação de objeto substituta. Esta implementação de objeto comunica-se com o servidor de registros para receber o estado da implementação de objeto falha e as requisições recebidas após o armazenamento deste estado. Finalmente, após o carregamento do estado e a re-execução das requisições, a nova implementação de objeto pode iniciar seus serviços.

6.4. Conclusões do Capítulo

A interceptação das chamadas aos objetos CORBA é prevista pela especificação CORBA. Todavia, tal interceptação deve ocorrer por meio do mecanismo de interceptores, também definido por esta especificação, e não através de mecanismos proprietários. Em particular, a funcionalidade dos interceptores consiste basicamente em ativar um ou mais serviços CORBA ou transparências. Além disso, múltiplos interceptores podem ser utilizados ao mesmo tempo oferecendo suporte simultâneo a diversas funcionalidades ou serviços.

Os ambientes Eternal e Phoinix permitem o desenvolvimento de aplicações confiáveis sobre *middlewares* CORBA por meio de funcionalidades ativadas a partir de

mecanismos de interceptação. No Phoinix, estes mecanismos consistem em estruturas geradas por um compilador IDL estendido a partir das definições de interface dos objetos CORBA. Já no Eternal, a interceptação das chamadas aos objetos CORBA é realizada por mecanismos do sistema UNIX. Neste contexto, nenhum dos ambientes apresentados utiliza a estrutura de interceptores definida pelo padrão CORBA.

O Phoinix oferece três níveis de serviços de tolerância a faltas, sendo que a confiabilidade do serviço aumenta a cada nível. Desta forma, o serviço de replicação, o único serviço Phoinix que apresenta suporte a grupos de objetos, possui maior capacidade de tolerância a faltas. Todavia, este serviço ainda não está implementado. Além disso, a definição das interfaces dos objetos tolerantes a faltas é feita por meio de uma extensão a linguagem OMG IDL. O Eternal permite o desenvolvimento de aplicações confiáveis por meio de suporte a grupos de objetos, possibilitando a replicação de clientes e servidores. Porém, o Eternal depende de uma ferramenta de comunicação em grupo proprietária, o Totem.

7 Estudo Comparativo entre as Abordagens

7.1 Introdução

Neste capítulo é apresentado um estudo comparativo entre as abordagens para o suporte a grupo sobre ORBs CORBA. Para tal, além de comentários a respeito destas abordagens, são apresentadas implementações desenvolvidas no contexto deste trabalho. Estas implementações representam as abordagens de serviço e interceptação, e além de ilustrar a utilização prática de conceitos CORBA, são utilizadas em avaliações de desempenho realizadas. Em particular, além destas implementações, é avaliado o Orbix+Isis. Neste sentido, são avaliados ambientes representando as três abordagens estudadas.

A seguir são apresentadas as implementações desenvolvidas. Posteriormente, é descrita uma avaliação qualitativa envolvendo os ambientes de suporte a grupo da literatura e as implementações deste trabalho. Finalmente, são apresentadas análises de desempenho.

7.2 Implementações Desenvolvidas

Conforme o que é dito anteriormente, as implementações desenvolvidas neste trabalho, representando as abordagens de serviço e interceptação, juntamente com o Orbix+Isis, representando a abordagem de integração, são utilizados para a confecção de análises de desempenho. Neste sentido, optou-se por utilizar serviços de suporte a grupo da ferramenta Isis para a confecção das implementações deste trabalho. Desta forma, todas as três abordagens avaliadas se utilizam desta ferramenta. Sendo assim, espera-se que os mecanismos de comunicação e gestão de grupo possuam uma sobrecarga semelhante nas três abordagens, permitindo que sejam avaliados apenas os seus processos de ativação.

Em particular, o desempenho das implementações das abordagens de serviço e de interceptação deve variar somente durante o estabelecimento das estruturas de grupo. Enquanto, na abordagem de serviço as próprias aplicações são responsáveis por requisitar a criação destas estruturas, na abordagem de interceptação esta criação é ativada transparentemente. Neste contexto, durante a invocação de um serviço prestado por um grupo de objetos, estas abordagens devem ter o mesmo desempenho.

Para o desenvolvimento das implementações foram utilizados a linguagem Java e o ORB OrbixWeb [IONA98]. Estas escolhas se fundamentam na produtividade da linguagem Java e no fornecimento de estruturas CORBA, como esqueletos e interfaces de invocação dinâmicos, pelo OrbixWeb. Além disso, apesar do OrbixWeb não oferecer os mecanismos de interceptores definidos na especificação CORBA, este ORB oferece mecanismos similares, chamados de filtros. Tais mecanismos, embora proprietários, são utilizados na confecção da abordagem de interceptação.

7.2.1. Implementação da Abordagem de Serviço

A implementação da abordagem de serviço é ilustrada na figura 7.1. Esta implementação prevê a justaposição de dois ambientes: o ORB e o suporte a grupo. Enquanto as comunicações ponto a ponto convencionais trafegam pelo ORB, as comunicações de grupo são implementadas usando estruturas de comunicação separadas. Estas estruturas separadas constituem o suporte a grupo, a interface de grupo e a ferramenta Isis.

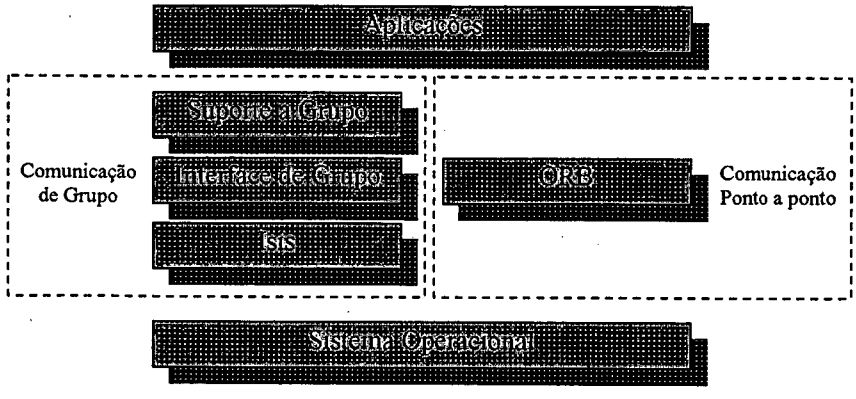


Figura 7.1 – Estrutura da implementação da abordagem de serviço

A ferramenta Isis oferece suporte de mais baixo para implementar os serviços de gestão e comunicação de grupo. O uso desta ferramenta, além de facilitar a implementação,

também oferece um maior desempenho, já que os mecanismos do Isis são extremamente otimizados. A interface de grupo visa livrar a implementação de características da ferramenta Isis. Esta interface consiste de objetos CORBA, os objetos de interface, que definem um padrão para o acesso aos serviços do Isis, desacoplando a implementação do suporte a grupo de detalhes do Isis. Finalmente, o suporte a grupo propriamente dito, consiste em um conjunto de objetos de serviço, chamados de *proxies*, que fornecem às aplicações uma interface para a utilização dos serviços de grupo. Na figura 7.2 é ilustrada a utilização de *proxies* e interfaces na difusão de mensagens.

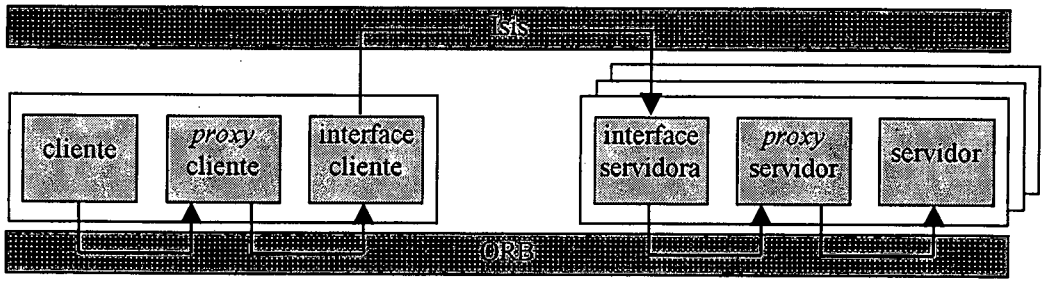


Figura 7.2 – Difusão de mensagem na implementação da abordagem de serviço

Devem ser criados objetos *proxies* e interfaces para cada grupo de qual um servidor deseja participar ou com o qual um cliente deseja comunicar-se. Estes objetos são criados a partir de fábricas. O uso de fábricas é obrigatório, já que a linguagem IDL não define construtores. Na figura 7.3 é ilustrada a criação de *proxies* e interfaces servidores. Inicialmente, o servidor da aplicação requisita a uma fábrica a criação de um *proxy*. Em seguida, este *proxy* transparentemente requisita a outra fábrica a criação de uma interface. A partir de então, o *proxy* e a interface criados podem ser utilizados normalmente pela aplicação. A criação destas estruturas no lado cliente é análoga.

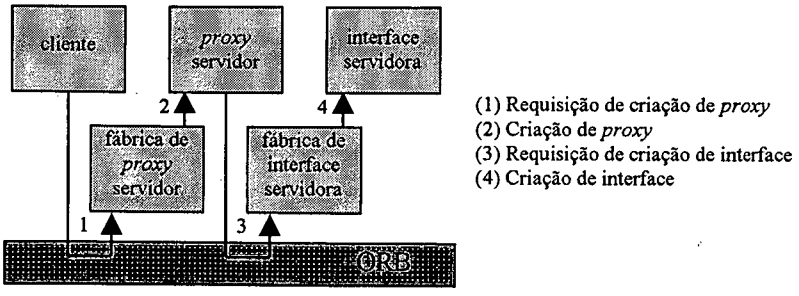


Figura 7.3 – Criação de *proxies* e interfaces servidores

Os *proxies* e interfaces devem ser criados em processos distintos, isto é, eles não devem residir no mesmo processo da fábrica. Isto se deve ao fato do Isis consistir em uma

ferramenta para grupos de processos Cada fábrica é responsável pela criação de apenas um único tipo de objeto. Além disso, cada fábrica deve responder por um domínio. Em particular, espera-se que esteja disponível uma fábrica para cada máquina da rede, o que garante que os *proxies* e interfaces não sejam objetos remotos, oferecendo um maior desempenho e confiabilidade às aplicações.

A função principal dos *proxies* é permitir a difusão de requisições dos clientes aos servidores. Esta tarefa é realizada por meio de esqueletos e interfaces de invocação dinâmicos. Em particular, os esqueletos dinâmicos permitem que os *proxies* clientes atendam requisições baseadas na interface do servidor. Por sua vez, as interfaces de invocação dinâmica permitem que os *proxies* servidores invoquem operações definidas nos membros de grupo. O uso destas estruturas é necessário, já que em tempo de compilação, não é possível determinar as operações do grupo ao qual os *proxies* serão associados.

```
module GroupService {

  // Group server member
  interface GroupServerMember {
    any getState ();
    void setState (in any state);
    void changeView (in GroupView view);
  };

  // Group server proxy
  interface GroupServerProxy {
    void joinGroup ();
    void leaveGroup ();
    GroupView getView ();
  };

  // Group server proxy factory
  interface GroupServerProxyFactory {
    GroupServerProxy create (in GroupServerMember groupServerMember, in string groupName);
    void destroy (in GroupServerProxy groupServerProxy);
  };

  // Group client proxy
  interface GroupClientProxy {
    GroupView getView ();
  };

  // Group client proxy factory
  interface GroupClientProxyFactory {
    GroupClientProxy create (in string groupName);
    void destroy (in GroupClientProxy groupClientProxy);
  };

};
```

Figura 7.4 – Interfaces IDL de membros de grupo, *proxies* e fábricas de *proxies*[1]

[1] Na descrição destas interfaces estão citadas apenas as operações mais relevantes. Em particular, as operações dos *proxies* invocadas pelas interfaces foram suprimidas.

No lado servidor os *proxies* muitas vezes devem invocar operações nos membros de grupo. Isto acontece durante as mudanças de visão do grupo e as transferências de estado. Neste contexto, os membros de grupo devem ser derivados de uma interface IDL padrão que define as operações invocadas. Na figura 7.4 é ilustrada esta interface, além das interfaces dos *proxies* clientes e servidores e suas fábricas.

A única operação disponível em um *proxy* cliente retorna a composição do grupo. Com esta informação as aplicações clientes podem executar comunicações ponto a ponto com cada membro do grupo. Além desta funcionalidade, os *proxies* servidores oferecem operações para a entrada e saída de membros de um grupo. A interface das fábricas permite apenas a criação e destruição de *proxies*. Finalmente, a interface dos membros de grupo permite a obtenção e configuração de seus estados, além da notificação de alterações na composição do grupo.

7.2.2. Implementação da Abordagem de Interceptação

A implementação da abordagem de interceptação permite que os mecanismos de grupo sejam transparentemente adicionados a aplicações clientes e servidoras. Esta transparência é alcançada a partir do uso de filtros do OrbixWeb. Estes filtros são associados às aplicações clientes e servidoras e se responsabilizam pela comunicação com as fábricas para a criação de *proxies*. Além disso, os filtros fazem com que a entrada e saída de membros de grupo seja automática. O OrbixWeb oferece dois tipos de filtros, os filtros de processo e de objeto. Os filtros de processo interceptam todas as requisições e respostas chegando ou partindo de um processo cliente ou servidor, independentemente dos objetos de origem e destino. Os filtros de objeto são aplicados a objetos individuais. Diferente dos filtros de processo, os filtros de objeto podem interceptar apenas requisições chegando e respostas partindo de um objeto servidor. Devido a esta limitação, foram utilizados filtros de processos na implementação da abordagem de interceptação.

Os filtros do OrbixWeb, embora funcionem de forma semelhante aos interceptores definidos pela especificação CORBA, possuem uma desvantagem. Estes filtros, diferente dos interceptores, não podem ser transparentemente associados aos objetos CORBA. Neste contexto, cada aplicação deve criar seu próprio filtro. Esta característica diminui um pouco

a transparência da abordagem de interceptação. Entretanto, a criação de um filtro ainda é bem mais simples do que o acesso direto a todos os serviços de grupo. Além disso, as aplicações também devem ter outra obrigação. Assim como na abordagem de serviço, os membros de grupo devem implementar uma interface IDL padrão que define as operações transferência de estado e notificação de mudança de composição do grupo.

Na abordagem de interceptação a utilização dos filtros é prevista somente durante a associação de um cliente ou um servidor a um grupo. Neste contexto, os filtros devem interceptar apenas as comunicações entre as aplicações e o serviço de nomes. Segundo a especificação CORBA, cada referência de objeto é associada a um ou mais nomes. O serviço de nomes é utilizado para registrar e recuperar estas referências. A abordagem de interceptação, aproveita esta característica do padrão CORBA para definir nomes de grupo. Um nome de grupo identifica um serviço que é prestado por um grupo de servidores. Cada nome de grupo é prefixado pela *string* "grp://". Esta característica visa facilitar a sua identificação. Em especial, um mesmo servidor, deve poder cadastrar-se individualmente ou como um membro de grupo. Desta forma, todas as requisições ao serviço de nomes que especificarem um nome de grupo devem ser interceptadas e manipuladas pelos filtros.

Para obter uma referência a um objeto servidor, os clientes realizam uma requisição ao serviço de nomes. Caso esta requisição referencie um nome de grupo, um filtro deve interceptá-la. Então, este filtro deve entrar em contato com uma fábrica para criar um *proxy* de grupo. Em seguida uma referência deste *proxy* é retornada ao cliente. Neste contexto, o cliente tem a ilusão de que recebeu uma referência para um objeto comum, quando na verdade recebeu uma referência para um *proxy*. Esta situação é ilustrada na figura 7.5.

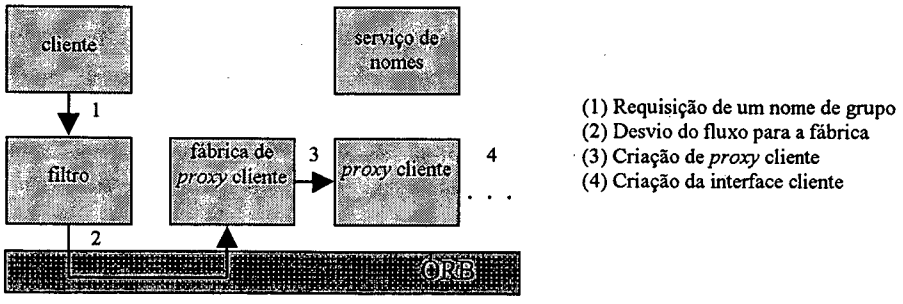


Figura 7.5 – Ativação das estruturas de grupo na abordagem de interceptação

Para cadastrarem-se, os servidores também devem realizar uma requisição ao serviço de nomes. Caso esta requisição especifique um nome de grupo, um filtro deve

interceptá-la. De modo similar ao lado cliente, este filtro deve entrar um contato com uma fábrica para criar um *proxy* de grupo. Além disso, o filtro deve invocar a operação de junção ao grupo no *proxy*. A partir de então, o servidor passa a receber requisições de seu *proxy* como se ele fosse um objeto cliente qualquer.

A implementação da abordagem de interceptação prevê que as aplicações clientes obtenham referências aos objetos servidores apenas utilizando o serviço de nomes. Todavia estas referências podem ser obtidas de outras formas, como em parâmetros de operações. Em particular, continua sendo permitido que uma aplicação cliente repasse uma referência de seu *proxy* para outra aplicação cliente. Entretanto, o acesso desta outra aplicação ao serviço prestado pelo grupo, pode ser realizado por um *proxy* remoto, comprometendo assim a confiabilidade do serviço. Além disso, uma aplicação servidora, ao invés de cadastrar-se no serviço de nomes, pode passar diretamente uma referência de si mesma para um cliente. Neste caso, o cliente acessará um serviço prestado por apenas um servidor e não por um grupo. Futuramente, espera-se contornar estes problemas, para tal todas as requisições que identificarem apenas um membro de grupo devem ser interceptadas e repassadas a todo o grupo. Ademais, sempre que uma referência a um *proxy* cliente for repassada a uma aplicação deve ser criado um novo *proxy* para esta aplicação.

7.3 Avaliação Qualitativa

A seguir é apresentada uma comparação qualitativa entre as três abordagens para o suporte a grupo sobre ORBs CORBA, discutindo critérios como: transparência, facilidade de uso, portabilidade, interoperabilidade e conformidade com o padrão CORBA. Esta comparação é baseada nas implementações originais das abordagens, isto é, o Orbix+Isis, Electra, OGS e Eternal. O OFS e o Phoinix são desconsiderados, já que os mecanismos para a tolerância a faltas oferecidos não suportam a comunicação em grupo. Além disso, eventualmente são tecidos comentários sobre as implementações desenvolvidas no contexto deste trabalho. Em particular, outras implementações, além das discutidas, podem apresentar características diferentes das levantadas.

7.3.1 Transparência

A transparência oculta o processamento em grupo do programador, dando a ilusão de que as invocações são originadas e atendidas por um objeto individual. Conforme é dito anteriormente neste trabalho, existem três tipos de transparência, ou seja, transparências de cliente, de servidor e de membro de grupo.

A transparência de cliente garante que os clientes enviem requisições e recebam respostas de um grupo servidor, como se este fosse um servidor único. A abordagem de integração (Orbix+Isis e Electra) apresenta este tipo de transparência, já que os clientes não precisam saber que as operações invocadas são atendidas por um grupo. Todavia, em certas situações, os clientes podem se beneficiar deste conhecimento. Nestas situações, os clientes recebem todas as respostas do grupo, podendo utilizar algoritmos especiais para selecionar a resposta correta. A abordagem de serviço (OGS) pode ser utilizada com ou sem transparência de cliente. No primeiro caso, os clientes invocam diretamente os serviços oferecidos pelo grupo. Isto é realizado por meio da utilização de esqueletos e interfaces de invocação dinâmicos durante a comunicação entre clientes e servidores. No segundo caso, clientes e servidores devem, respectivamente, invocar e atender operações específicas para a comunicação de grupo. A abordagem de interceptação (Eternal) obriga a transparência de cliente. Diferente das outras abordagens, um cliente não pode acessar todas as respostas de uma invocação. Em particular, as implementações das abordagens de serviço e interceptação desenvolvidas neste trabalho, também possuem transparência de cliente. Da mesma forma do que acontece com o OGS, esta transparência é garantida pela utilização de esqueletos e interfaces de invocação dinâmicos.

A transparência de servidor garante que um grupo cliente possa interagir com um servidor da mesma forma que um cliente único o faria. Este tipo de transparência só é possível caso haja suporte a replicação de clientes. Neste sentido, apenas a abordagem de interceptação (Eternal) apresenta transparência de servidor.

Finalmente, a transparência de membro de grupo garante que os membros possam se comportar como se não fizessem parte de um grupo. Desta forma, os membros não podem diferenciar comunicações individuais de mensagens enviadas a todo o grupo. Em particular, este tipo de transparência é suportado pelas três abordagens.

7.3.2 Facilidade de Uso

A facilidade de uso é uma consideração importante, já que diminui o tempo de desenvolvimento e manutenção dos programas, tornando-os mais robustos e confiáveis. Por facilidade de uso entende-se a baixa complexidade na ativação e utilização dos mecanismos de gestão e comunicação de grupo. Neste contexto, esta característica está altamente relacionada com a transparência destes mecanismos.

As abordagens de integração (Orbix+Isis e Electra) e interceptação (Eternal) apresentam maior facilidade de uso, já que o estabelecimento das estruturas de grupo é automático. Na abordagem de serviço (OGS), este estabelecimento é ativado pelos objetos clientes e servidores. Neste sentido, a configuração do suporte a grupo é explícita, mas a comunicação em grupo pode ser transparente. Desta forma, esta abordagem combina mecanismos de configuração flexíveis com suporte a grupo transparente.

No que diz respeito as implementações desenvolvidas neste trabalho, também é claro que a implementação da abordagem de interceptação apresenta maior facilidade de uso do que a implementação da abordagem de serviço. Enquanto na abordagem de serviço, a criação e ativação das estruturas de suporte a grupo deve ser iniciada pelos objetos de aplicação, na abordagem de interceptação esta tarefa é realizada automaticamente por meio dos mecanismos de filtros. Todavia, é importante ressaltar que uma vez ativadas as estruturas de grupo, a sua utilização é transparente nas duas implementações. Neste contexto, a comunicação em grupo não possui nenhuma complexidade adicional em relação a uma comunicação ponto a ponto convencional.

7.3.3 Portabilidade

A portabilidade de um código implica na independência deste código de um ORBs específicos. Em particular, são consideradas a portabilidade do código do suporte a grupo e das aplicações desenvolvidas sobre este suporte.

Na abordagem de integração (Orbix+Isis e Electra), o código dos mecanismos de suporte a grupo é integrado ao ORB. Além disso, as aplicações desenvolvidas utilizam construções não padronizadas pelo CORBA. Neste sentido, os códigos do suporte e das aplicações não são portáveis. Na abordagem de serviço (OGS), tanto o código das

aplicações como do suporte a grupo são portáveis, já que não dependem de características da implementação de um ORB específico. Finalmente, na abordagem de interceptação (Eternal), são utilizados mecanismos do Unix para suportar grupos. Desta forma, os mecanismos de grupo desta abordagem não são portáveis. Todavia, estes mecanismos não são referenciados no código das aplicações, tornando-as completamente portáveis.

No que diz respeito as implementações desenvolvidas neste trabalho, o código das aplicações apresenta portabilidade. No caso da abordagem de interceptação, os mecanismos de grupo são ativados transparentemente por filtros, e no caso da abordagem de serviço, estes mecanismos são ativados por meio de operações de objetos de serviço definidos por uma interface IDL. Entretanto, o código de suporte a grupo das implementações desenvolvidas não apresenta portabilidade, já que são utilizadas facilidades da ferramenta Isis para a sua implementação. Além disso, na abordagem de interceptação, são utilizados mecanismos de filtros, que embora se assemelhem aos mecanismos interceptores definidos pela especificação CORBA, consistem em soluções proprietárias.

7.3.4 Interoperabilidade

A interoperabilidade implica na possibilidade de aplicações em ORBs distintos interagirem. Em particular, a interação entre estas aplicações pode ser parcial, isto é, envolvendo apenas comunicações ponto a ponto, ou completa, isto é, também envolvendo comunicações em grupo.

Implementações que fazem uso de sistemas de comunicação proprietários não são interoperáveis. Este é o caso do Electra. O Orbix+Isis combina invocações de grupo sobre o Isis e invocações ponto a ponto sobre o protocolo IIOP. Desta forma, o Orbix+Isis é parcialmente interoperável. O Eternal pode escolher quais requisições devem ser interceptadas e repassadas ao Totem, também podendo interoperar sobre o IIOP. A abordagem de serviço (OGS) utiliza apenas primitivas de comunicação do ORB, sendo completamente interoperável.

As implementações desenvolvidas neste trabalho prevêem que enquanto as comunicações em grupo sejam transmitidas pela ferramenta Isis, as comunicações ponto a

ponta sejam transmitidas pelos mecanismos convencionais do ORB. Desta forma, estas implementações consistem em soluções parcialmente interoperáveis.

7.3.5 Conformidade com o Padrão CORBA

A abordagem de integração (Orbix+Isis e Electra) não está em conformidade com o padrão CORBA, já que as referências de objeto podem identificar grupos, ao invés de somente objetos individuais. Além disso, as regras de mapeamento das estruturas definidas em IDL para a linguagem de implementação foram estendidas pelo Orbix+Isis para gerar códigos de gestão e comunicação de grupo. Ademais, apesar do Electra respeitar as regras de mapeamento, classes da implementação do ORB foram estendidas para permitir o suporte a grupo.

As abordagens de serviço (OGS) e interceptação (Eternal) respeitam a especificação CORBA. Os mecanismos de grupo na abordagem de serviço são independentes do núcleo do ORB, consistindo em objetos de serviço e suas interfaces IDL. A abordagem de interceptação é completamente independente do ORB, sendo baseada em estruturas do protocolo IIOP. Todavia, apesar desta abordagem não estender nenhum mecanismo do CORBA, ela não segue a filosofia da OMG, que prevê que implementações de ORBs devam ser completamente interoperáveis e portáveis.

As implementações desenvolvidas neste trabalho também estão em conformidade com o padrão CORBA, não estendendo nenhum mecanismo desta especificação. Todavia, da mesma forma que o Eternal, estas implementações são dependentes de uma ferramenta de comunicação proprietária.

7.4. Avaliações de Desempenho

As avaliações de desempenho foram realizadas com base em testes envolvendo três estações Ultra Sparc rodando o sistema operacional Solaris 2.5 sobre uma rede local Ethernet de 10 Mbs. O limite no número de estações se deve a condições da licença da ferramenta Isis disponível. Nestas avaliações de desempenho também deve ser considerado que enquanto no Orbix+Isis, as aplicações são desenvolvidas em C++, nas implementações deste trabalho,

as aplicações são desenvolvidas em Java. A linguagem Java 1.1 apresenta um desempenho cerca de dez vezes menor que a linguagem C [Flanagan97]. Todavia, o impacto do uso da linguagem Java é bem menor, já que os mecanismos de difusão e gestão de grupo são providos pelo Isis. Além disso, um dos fatores mais importantes para o desempenho é o tempo de transmissão na rede.

Na figura 7.6 é apresentado o tempo médio de resposta no Orbix+Isis e nas implementações das abordagens de serviço e interceptação a uma invocação de um serviço prestado por um grupo de objetos. A média foi obtida a partir do tempo de resposta das primeiras mil invocações. Além disso, é ilustrada a evolução dos tempos de resposta de acordo com o número de servidores presentes no grupo. Cada servidor é executado em uma máquina distinta. Como pode ser percebido, o Orbix+Isis possui um desempenho superior. Embora com o aumento do número de servidores presentes no grupo exista um aumento no tempo de resposta, a diferença de desempenho entre os ambientes permanece constante. Neste contexto, caso o número de servidores seja muito grande, espera-se que a diferença de desempenho seja desprezível. A justificativa para a diferença de desempenho entre as abordagens é a utilização de comunicações ponto a ponto do ORB entre os objetos intermediários nas implementações das abordagens de serviço e interceptação.

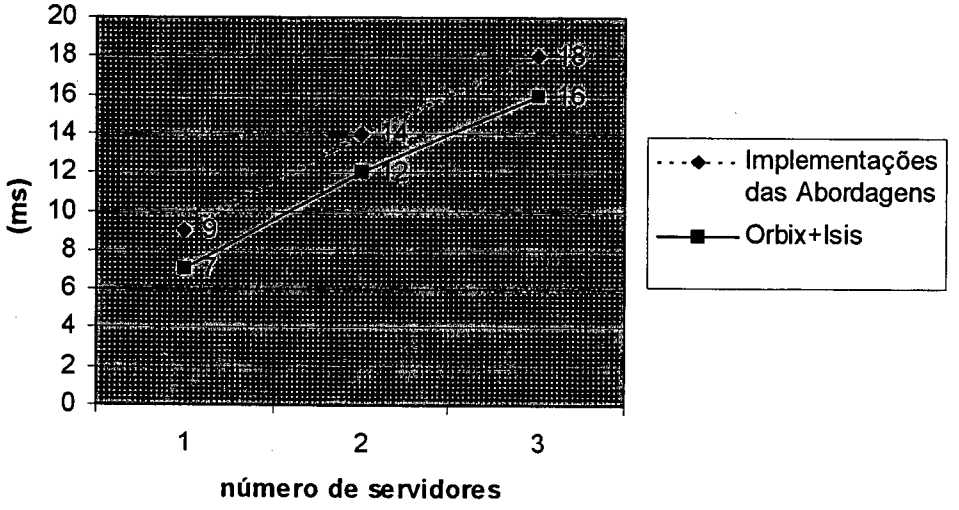


Figura 7.6 – Tempo médio de resposta de acordo com o número de servidores

A fim de avaliar melhor o desempenho das implementações das abordagens, foram levantados tempos de resposta parciais a uma invocação de um serviço prestado por um grupo de objetos. Conforme o ilustrado na figura 7.7, os tempos de resposta parciais envolvem o tráfego entre os processos participantes do serviço de grupo. Neste tráfego são considerados o tempo para o envio de uma requisição e o retorno de seu resultado.

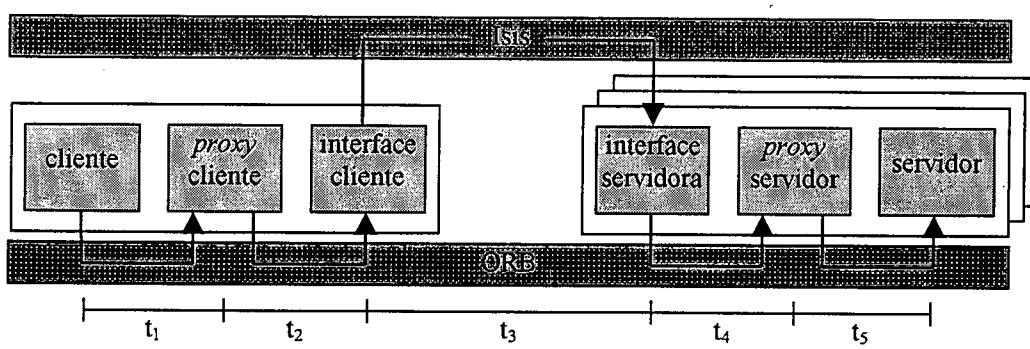


Figura 7.7 – Definição dos tempos de resposta parciais

Na figura 7.8(a) é apresentado o impacto de cada tempo parcial na execução de uma invocação a um serviço prestado por um grupo de servidores. Pode ser percebido que o aumento do número de servidores implica apenas no aumento no tempo de tráfego entre as interfaces clientes e servidoras. Esta característica é esperada, já que a comunicação entre as aplicações e os *proxies* e entre os *proxies* e as interfaces, envolve apenas tráfego local. Além disso, é somente na comunicação entre as interfaces que ocorre a execução da difusão em grupo. Em particular, os tempos t_1, t_2, t_4 e t_5 se devem a sobrecargas da comunicação entre os objetos CORBA, enquanto o tempo o tempo t_3 envolve a execução da ferramenta Isis. A seguir são apresentadas alternativas de implementação das abordagens que visam o aumento de desempenho. Estas alternativas visam a diminuição do tempo envolvido nas comunicações locais, isto é, busca-se evitar as sobrecargas da comunicação entre os objetos CORBA.

A primeira alternativa de implementação envolve a eliminação dos processos onde residem as interfaces. Neste contexto, ao invés de entrar em contato com uma fábrica, os *proxies* instanciam diretamente as interfaces. Desta forma, os *proxies* e interfaces deixam de residir em processos distintos, fazendo com que a interação entre estas estruturas passe a ser executada em um mesmo espaço de endereçamento. Esta alternativa não causa nenhum impacto sobre as aplicações clientes e servidoras, já que estas aplicações mantém a mesma interface com os *proxies*. Neste contexto, esta alternativa pode ser implementada segundo

as abordagens de serviço e interceptação. Todavia, a implementação dos *proxies* passa a conter código para o acesso a ferramenta Isis, diminuindo a sua adaptabilidade. Em particular, as interfaces clientes e servidoras deixam de ser consideradas como objetos CORBA, para serem consideradas objetos ordinários de uma aplicação, fazendo com que a sua interação com os *proxies* seja mais eficiente. Na figura 7.8(b) são apresentados tempos médios de reposta desta alternativa. O tempo t_{234} envolve as comunicações do *proxy* cliente ao *proxy* servidor e vice-versa. Segundo estas avaliações, pode-se perceber que houve uma considerável diminuição nos tempos de resposta.

	1	2	3
t_1	0,5	0,5	0,5
t_2	0,5	0,5	0,5
t_3	7	12	16
t_4	0,5	0,5	0,5
t_5	0,5	0,5	0,5
t	9	14	18

(a) Implementação original

	1	2	3
t_1	0,5	0,5	0,5
t_{234}	7,5	12,5	16,5
t_5	0,5	0,5	0,5
t	8,5	13,5	17,5

(b) Ausência de fábrica de interfaces

	1	2	3
t	8	13	17

(c) Ausência de fábricas

Figura 7.8 – Tempos médios de resposta parciais nas implementações desenvolvidas

A segunda alternativa de implementação envolve tanto a eliminação dos processos onde residem as interfaces, como a eliminação dos processos onde os *proxies* residem. De forma semelhante à alternativa anterior, as aplicações clientes e servidoras passam a instanciar diretamente os *proxies*. Deste modo, estes *proxies* não são mais definidos por interfaces IDL, tornando-se objetos ordinários de uma aplicação. Esta alternativa, conforme ilustrado na figura 7.8(c), embora também apresente um considerável aumento de desempenho, eliminando toda a sobrecarga de comunicação entre os objetos CORBA intermediários e aproximando-se do alcançado pelo Orbix+Isis, possui algumas desvantagens. Esta alternativa não consiste em uma solução CORBA, já que o código dos *proxies* é disponibilizado às aplicações como parte de uma biblioteca. Além disso, a sua implementação não consiste nem na abordagem de serviço, nem na abordagem de interceptação.

Outro fator de desempenho que deve ser observado é o tempo necessário para o estabelecimento das estruturas de grupo, isto é, os *proxies* e interfaces. Na implementação

da abordagem de serviço este tempo é de 2 ms, já na abordagem de interceptação são perdidos 3 ms. Esta diferença se deve ao uso de filtros na abordagem de interceptação. No caso do Orbix+Isis, as estruturas de grupo são inerentes a qualquer aplicação. Desta forma, não é perdido tempo para a sua criação. Todavia, espera-se que mesmo as aplicações que não envolvam grupos possuam uma pequena sobrecarga.

7.5 Conclusões do Capítulo

Um serviço de suporte a grupo pode ser implementado tanto no topo de um sistema de comunicação de grupo específico, bem como, sobre os mecanismos de comunicação do ORB [Felber96]. Todavia, a última abordagem parece ser mais interessante, já que o uso de um canal separado para as comunicações de grupo implicaria em uma solução proprietária. Uma potencial desvantagem do uso dos mecanismos de comunicação do ORB é o comprometimento do desempenho, uma vez que as comunicações são ponto a ponto. Entretanto, a OMG recomenda assumir-se implementações de CORBA eficientes quando projeta-se um serviço [OMG95b].

Diferente do OGS, a implementação da abordagem de serviço apresentada neste trabalho, depende de um sistema de comunicação em grupo proprietário, ou seja, o Isis. Esta escolha, além de facilitar o desenvolvimento da implementação, visa auxiliar as análises de desempenho. Neste sentido, os mecanismos de gestão e comunicação de grupo nas três abordagens são baseados no Isis, devendo apresentar uma sobrecarga semelhante e permitindo que sejam avaliados apenas os seus processos de ativação. Futuramente, a ferramenta Isis deve ser substituída por objetos de serviço, limitando as comunicações a somente o ORB. Este objetivo é previsto no projeto GROUPAC [Lung99], que visa a implementação de um suporte de soluções abertas para aplicações confiáveis

A implementação da abordagem de interceptação apresentada também difere do proposto em [Narasimhan97a]. Diferente do Eternal, onde a interceptação é realizada por mecanismos do sistema operacional Unix que capturam as mensagens IIOP, na implementação da abordagem de interceptação são utilizados filtros do OrbixWeb. A escolha destes filtros deve-se a sua similaridade com o mecanismo de interceptores

definidos na especificação CORBA. A implementação da abordagem de interceptação deve passar a utilizar estes mecanismos assim que forem introduzidos em ambientes CORBA. O uso de outras estruturas, além dos interceptores, implica em uma solução proprietária.

A especificação CORBA define os interceptores como estruturas que ativam transparentemente um ou mais serviços [OMG95b]. Neste sentido, qualquer serviço CORBA pode ser invocado diretamente pelas aplicações ou ativado por um interceptor. A implementação da abordagem de interceptação segue esta diretriz. Desta forma, a abordagem de interceptação consiste apenas em uma evolução da abordagem de serviço.

8. CONCLUSÃO

Neste trabalho são discutidas várias soluções para a adição de mecanismos de processamento de objetos em grupo ao padrão CORBA. Estas soluções podem ser agrupadas em três abordagens. Na abordagem de integração o núcleo do ORB é alterado para permitir a execução dos mecanismos de processamento em grupo. Na abordagem de serviço, estes mecanismos são disponibilizados como objetos de serviço e suas interfaces IDL. Finalmente, na abordagem de interceptação, o processamento em grupo é ativado transparentemente por meio de estruturas de reflexão computacional. Em particular, apenas as abordagens de serviço e interceptação são previstas pelo padrão CORBA. Todavia, para apresentarem portabilidade e interoperabilidade, as implementações destas abordagens não devem utilizar sistemas de comunicação em grupo proprietários. Ademais, na abordagem de interceptação, devem ser utilizados os mecanismos de interceptores definidos pelo padrão CORBA.

Em especial, a abordagem de interceptação pode ser considerada como uma evolução da abordagem de serviço. Neste sentido, uma implementação da abordagem de interceptação pode consistir apenas na captura de chamadas aos objetos CORBA e o seu desvio a um ou vários objetos de serviço implementando as funcionalidades de suporte a grupo. Desta forma, as abordagens de interceptação e de serviço diferenciam-se apenas devido a que na primeira abordagem, os mecanismos de grupo são ativados transparentemente.

Além disso, neste trabalho também são desenvolvidas implementações, representando as abordagens de serviço e interceptação, sobre as quais são realizadas análises de desempenho. Estas avaliações de desempenho apontaram uma sobrecarga mínima em relação à abordagem de integração, representada pelo Orbix+Isis. Todavia, as implementações desenvolvidas são baseadas em mecanismos de difusão e gestão de grupo fornecidos pela ferramenta Isis. Esta característica fere a especificação CORBA, consistindo

em uma solução proprietária. Desta forma, futuramente pretende-se desenvolver uma solução completamente aberta. Neste contexto, a gestão e a difusão em grupo devem ser desenvolvidas somente a partir de objetos CORBA. Ademais, atualmente as implementações abordam apenas a replicação de servidores, não oferecendo primitivas para a comunicação entre grupos e portanto, não suportando replicação de clientes. Em particular, tanto os conhecimentos adquiridos em seu projeto, como as próprias implementações criadas, com as devidas modificações, devem ser utilizados posteriormente no projeto GROUPAC [Lung99], que visa a implementação de uma solução aberta para o suporte a aplicações confiáveis.

9. REFERÊNCIAS BIBLIOGRÁFICAS

[Amir92] AMIR, Y. & DOLEV, D & KRAMER, S. & MALKI, D. *Transis: A Communication Subsystem for High Availability.* In 22nd International Symposium on Fault-Tolerant Computing, IEEE, julho de 1992.

[Amir95] AMIR, Y. & MOESER, L. E. & MELLIAR-SMITH, P. M. & AGARWAL, D. A. & CIARFELLA, P. *The Totem Single-ring Ordering and Membership Protocol.* ACM Transactions on Computer Systems, 13(4):311-342, novembro de 1995.

[Birman90] BIRMAN, K. & COOPER, R. & JOSEPH, T. A. & MARZULLO, K. MAKPANGOU, M. & KANE, K. & SCHMUCK, F. & WOOD, M. *The Isis System Manual.* Department of Computer Science, Cornell University, setembro de 1990.

[Birman91] BIRMAN, Kenneth P. & SCHIPER, André & STEPHENSON, Pat. *Lightweight Causal and Atomic Group Multicast.* ACM Transactions on Computer Systems, vol. 9, no. 3, agosto de 1991.

[Birman93] BIRMAN, K. P. *The Process Group Approach to Reliable Distributed Computing.* Communication of ACM, vol. 36, n. 12, 1993, pp 37-53.

[Chen92] CHEN, I. R. & BASTINI, F. B. *Reliability of fully and partially replicated systems*, IEEE Trans. Reliability, vol 41, nº 2, pp. 175-182, junho de 1992.

[Chorus92] CHORUS Systèmes, *Chorus Simulator v3 r4: Programmer's Guide*, 1992.

[Cristian95] CRISTIAN, F. & SCHMUCK, F. *Agreeing on Processor Group Membership in Assyncronous Distributed Systems*", Technical Report CSE95-428, University of California at San Diego, EUA, 1995.

[Fabre95] FABRE, Jean-Charles & NICOMETTE, Vincent & PÉRENNOU, Tanguy & STROUD, Robert J. & WU, Zhixue. *Implementing Fault Tolerant Aplications using Reflexive Object-Oriented Programming.* 25th IEEE International Symposium on Fault-Tolerant Computing, Passadena-CA, pp. 488-498, junho de 1995.

[Felber96] FELBER, Pascal & GARBINATO, Benoît & GUERRAOUI, Rachid. *The Design of a CORBA Group Communication Service.* 15th Symposium on Reliable Distributed Systems, pp 150-159, outubro de 1996.

[Felber97] FELBER, Pascal & GUERRAOUI, Rachid & SCHIPER, André. *A CORBA Object Group Service.* Ecole Polytechnique Fédérale de Lausanne, Computer Science Department, Technical Report, 1997

[Ferber89] FERBER, J. *Computational Reflection in class based Object-Oriented Languages.* Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'89, New Orleans-LA, pp. 317-326, outubro de 1989.

[Flanagan97] FLANAGAN, David. *Java in a Nutshell.* Editora O'Reilly, 2ª edição, maio de 1997.

[Fraga97] FRAGA, Joni & MAZIERO, Carlos & LUNG, Lau C. & LOQUES FILHO, Orland G. *Implementing Replicated Services in Open Systems Using a Reflexive Approach.* 3rd International Symposium on Autonomous Decentralized Systems, pp 273-280, Berlim - Alemanha, abril de 1997.

[Hagsand91] HAGSAND, Olof. *ISIS/C++ Reference Manual and Tutorial.* Isis Distributed Systems, Inc., novembro de 1991.

[Hagsand92] HAGSAND, Olof & HERZOG, Holger & BIRMAN Kenneth & COOPER, Robert. *Object Groups: An Approach to Reliable Distributed Programming*. Cornell University, submetido a ECOOP'92.

[Huang93] Huang, Y. & Kintala C. *Software Implemented Fault Tolerance*, in Proc. of 22nd Fault-tolerance Computing Symposium, pp. 2-10, 1993.

[IONA95] IONA Technologies, Ltd. *Orbix Programming Guide*, novembro, 1995.

[IONA98] IONA Technologies PLC. *OrbixWeb Programmer's Guide*, novembro de 1997.

[Isis93] ISIS Distributed Systems, Inc. *Object Groups: A response to the ORB 2.0 RFI*, abril de 1993.

[Isis94] ISIS Distributed Systems, Inc. *RDO/C++: Tutorial Release 1.0.3*, abril de 1994.

[Isis95] ISIS Distributed Systems, Inc. & IONA Technologies, Ltd. *Orbix+Isis Programmer's Guide*, Document D070-00, 1995.

[ISO93] ISO/IEC 10746-3. *ITU-TS Recommendation X.903: Basic Reference Model for Open Distributed Processing: Prescriptive Model*. 2nd CD draft, junho de 1993.

[Kaashoek92] KAASHOEK, M. F. & TANENBAUM, A. S. *Efficient Reliable Group Communication for Distributed Systems*. Rapport IR-295, Faculteit Wiskunde en Informatica, Vrije Universiteit, julho de 1992.

[Keahey98] KEAHEY, Kate. *A Brief Tutorial on CORBA*. Disponível em http://www.cs.indiana.edu/hyplan/kksiazek/tuto.html [12 de janeiro de 1998].

[Lea94] LEA, Doug. *Objects in Groups*. SUNY at Oswego / NY Case Center, submetido a ECOOP'94.

[Liang90] LIANG, Luping & CHANSON, Samuel T. & NEUFELD, Gerald W. *Process Groups and Group Communications: Classifications and Requirements*. IEEE Computer, pp 56-65, fevereiro de 1990.

[Liang96] LIANG, Deron & CHOU, S. C. & YUAN, S. M. *Phoinix: A Fault-Tolerant Object Service in OMA*. 1996.

[Liang98] LIANG, Deron & FANG, Chen-Liang. & YUAN, Shyan-Ming. *A Fault-Tolerant Object Service on CORBA*. submetido a Journal of Systems and Software, 1998.

[Lisbôa97] LISBÔA, Maria Lúcia. *Arquiteturas de Meta-nível*, Simpósio Brasileiro de Engenharia de Software SBES'97, Fortaleza-CE, Brasil, 1997.

[Lung96] LUNG, Lau Cheuk. *Implementação de Técnicas de Replicação de Componentes de Software sobre a Plataforma Aberta CORBA*. Dissertação de Mestrado, LCMI/UFSC, 1996.

[Lung99] LUNG, Lau & FRAGA, Joni da Silva & FARINES, Jean-Marie. *CosNamingFT – Um Serviço de Nomes Tolerante a Falhas em Conformidade com o Padrão OMG*. Submetido ao SBRC 1999.

[Maes87a] MAES, P. *Computational Reflection*, PhD thesis, Vrije University, Brussel, 1987, also Technical Report 87-2.

[Maes87b] MAES, P. *Concepts and Experiments in Computational Reflection*, in Proc. OOPSLA 87, pp. 147-155, 1987.

[Maffeis93] MAFFEIS, Silvano. *Distributed Programming using Object-Groups*. Dept. of Computer Science, University of Zurich, Technical Report Tr. 93.38, 1993.

[Maffeis95a] MAFFEIS, Silvano. *Adding Group Communication and Fault Tolerance to CORBA*. USENIX Conference on Object-Oriented Technologies, junho 1995.

[Maffeis95b] MAFFEIS, Silvano. *An Interoperable Middleware Infrastructure Supporting Object-Group Communication*. SIGCOMM Workshop Position Paper, 1995.

[Maffeis96] MAFFEIS, Silvano. *Electra 2.0b Programmer's Manual*. Dept. of Computer Science, Cornell University, 1996.

[Mishra93] MISHRA, S. & PETERSON, L. L. & SCHLICHTING, R. D. *Consul: A Communication Substrate for Fault-Tolerant Distributed Programs*. Distributed Systems Engineering Journal 1,2, dezembro de 1993.

[Nacamura96] NACAMURA Júnior, Luiz. *Proposta de um Esquema de Tolerância a Faltas baseado em Múltiplas Replicações de Processos: Esquema MR*. Tese de Doutorado, LCMI/UFSC, 1996.

[Narasimhan97a] NARASIMHAN, P. & MOSER L. E. & MELLIAR-SMITH P. M. *Exploiting the Internet Inter-ORB Protocol Interface to Provide CORBA with Fault Tolerance*. Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems, junho de 1997.

[Narasimhan97b] NARASIMHAN, P. & MOSER L. E. & MELLIAR-SMITH P. M. *Replica consistency of CORBA objects in partionable distributed systems*. Distributed Systems Engineering, vol. 4, pp. 1-12, setembro de 1997.

[OMG94] OMG. *IDL C++ Language Mapping Specification*. OMG Document 94-9-14, 1994.

[OMG95a] OMG. *The Common Object Request Broker: Architecture and Specification*. OMG, 1995.

[OMG95b] OMG. *CORBAservices: Common Object Services Specification*. OMG, 1995.

[OMG97] OMG. *A Discussion of the Object Management Architecture*. OMG, janeiro de 1997.

[OMG98a] OMG. *Portable Interceptor RFP- Request for Proposal OMG Document Orbos/ 98-05-05*, maio de 1998.

[OMG98b] OMG. *Fault-Tolerant CORBA Using Entity Redundancy*. RFP orbos/98-04-01, October, 1998.

[Oskiewicz93] OSKIEWICZ E. & EDWARDS N. *A Model for Interface Groups*. Technical Report AR.002.01, ANSA, Architecture Projects Management Limited, Cambridge UK, 1993.

[Powell91] POWELL, D. "*Delta-4 Architecture Guide*", Esprit II P2252, Delta-4 Phase 3, agosto de 1991.

[Renesse93a] RENESSE, R. Van & BIRMAN, Kenneth P. & COOPER, R. *The HORUS System*. Technical Report, Cornell University, 1993.

[Renesse93b] RENESSE, R. Van. *A MUTS Tutorial*. Muts Documentation, Cornell University, 1993.

[Rosenberry92] ROSENBERRY, W. & KENNEY, D. & FISHER, G. *Understanding DCE*. O'Reilly & Associates, Inc., 1992.

[Schlichting83] SCHLICHTING, R. D. & SCHNEIDER, F. B. *Fail_Stop Processors: An approach to designing fault-tolerant computing systems*. ACM Transactions on Computer Systems, vol. 1, no. 3, pp. 222-238, agosto de 1983.

[Schmidt97] SCHMIDT, Douglas C. *Overview of CORBA*. Novembro 1997. Disponível em http://www.cs.wustl.edu/~schmidt/corba-overview.html [12 de janeiro de 1998].

[Shrivastava92] SHRIVASTAVA, S. K. & EZHILCHELVAN, P.D. & SPEIRS, N. A. & SEALTON D. T. *Fail-Controlled Computer Architecture for Distributed Systems*, Technical Report n. 333, University of Newcastle upon Type, agosto de 1992.

[Smith82] SMITH, Brian C. *Reflection and Semantics in Procedural Languages*, PhD thesis, M.I.T., 1982.