Obtendo interoperabilidade entre ORBs

*Nuccio M. S. Zuquello** *Edmundo R. M. Madeira*†

Departamento de Ciência da Computação
Universidade Estadual de Campinas
13081-970 Campinas, São Paulo
E-mail: {zuquello,edmundo}@dcc.unicamp.br

Resumo

A capacidade de sistemas heterogêneos cooperarem é uma necessidade atual que vem se impondo cada vez mais, principalmente pelo rápido desenvolvimento da tecnologia de processamento distribuído, obrigando a busca de soluções mais apropriadas. Nesse contexto, a especificação CORBA (*Common Object Request Broker Architecture*) apresenta uma arquitetura que permite a interoperabilidade entre aplicações em ambientes distribuídos heterogêneos. Entretanto, pela sua grande flexibilidade nas decisões de implementação, surgem problemas para estender essa interoperabilidade entre duas ORBs desenvolvidas por diferentes tecnologias. Este artigo discute esses problemas e as respectivas soluções, e apresenta um conjunto de operações para suportar o mecanismo de interceptação, estendendo as transparências de acesso e localização, consideradas fundamentais num ambiente distribuído, para além do escopo de uma ORB. É proposta, também, uma forma para prover transparência de relocação, estendida para suportar interoperabilidade. Por fim, é descrita uma implementação desses mecanismos utilizando um protótipo de ORB, desenvolvido na UNICAMP, e uma implementação comercial, chamada ORBeline.

Abstract

The ability of heterogeneous systems cooperate to problem's solutions is a real need that has been growing, due to the increasing development of the distributed processing technology. This need is driving the development of more suitable solutions. The CORBA (*Common Object Request Broker Architecture*) specification presents an architecture which allows interoperability among heterogeneous distributed environment applications. However, because of the great flexibility offered in implementation decisions, many problems to extend interoperability between two ORBs developed by different technologies arise. This paper discusses these problems and their solutions and presents a set of operations to support an interception mechanism extending access and location transparencies, considered as fundamental for a distributed environment, to work beyond an ORB scope. A scheme to provide relocation transparency, extended to support interoperability is also proposed. Finally, an implementation of these mechanisms over an ORB's prototype, developed at UNICAMP, and ORBeline, a commercial product, is described

*Mestrando do Departamento de Ciência da Computação - UNICAMP.
†Professor do Departamento de Ciência da Computação - UNICAMP.

1 Introdução

A contínua tendência ao rápido crescimento do processamento distribuído ajusta-se à realidade atual, onde a informação pode ser armazenada e processada em pontos distintos, geograficamente separados e com grande heterogeneidade de meios [4]. O desenvolvimento tecnológico, possibilitando a produção de máquinas menores e mais poderosas, também colabora para essa distribuição [5].

Nesse contexto, a orientação a objetos parece ser a abordagem mais apropriada ao desenvolvimento da tecnologia para suportar a computação distribuída, oferecendo boas soluções para problemas originados pela heterogeneidade, gerenciamento de configuração, contabilidade, monitoramento, etc, os quais tornam-se críticos numa distribuição em larga escala. Essa abordagem está sendo utilizada, por exemplo, pela ISO, ITU-T, APM (*Architecture Projects Management*) e OSF (*Open Software Foundation*) [4, 5, 17].

Visando uma padronização para as plataformas que pretendem solucionar os problemas do processamento distribuído, a ISO, juntamente com a ITU-T, vem desenvolvendo um Modelo de Referência para ODP (*Open Distributed Processing*) [3].

Dentre as várias plataformas existentes, a CORBA (*Common ORB Architecture*) afigura-se como uma das mais adequadas [21], possuindo os conceitos básicos definidos pelo RM-ODP. A especificação dessa plataforma, entretanto, permite um alto grau de liberdade na sua implementação. Dessa forma, duas ORBs desenvolvidas por diferentes fabricantes não conseguem interoperar no estado atual das implementações: um cliente, numa ORB, não consegue solicitar serviços fornecidos por um objeto em outra ORB.

Neste trabalho serão descritas as necessidades inerentes à capacidade de interoperabilidade [2, 13], sendo apresentadas algumas soluções com suas propostas de implementação, sendo mantidas as transparências de acesso e localização da ORB. Além disso, é proposto um mecanismo para suporte à transparência de relocação, estendido para interoperabilidade [2]. Para a implementação é utilizado uma plataforma comercialmente disponível, chamada ORBeline [18], e um protótipo de ORB desenvolvido na UNICAMP [1].

Este trabalho faz parte de um projeto, chamado de Multiware [2, 6, 23], para suporte a aplicações distribuídas, em desenvolvimento na UNICAMP, e na qual se pretende incorporar os conceitos definidos no RM-ODP usando, como plataforma inicial, uma ORB.

Na seção 2 serão apresentados os conceitos básicos da CORBA. A seção 3 identifica as necessidades decorrentes do suporte à interoperabilidade, expõe algumas soluções e propõe uma forma de implementá-las. A seção 4 descreve uma forma de prover transparência de relocação entre ORBs interoperantes. Uma implementação dos mecanismos apresentados nas seções anteriores é descrita na seção 5. A seção 6 resume outros trabalhos relacionados e segue-se a conclusão na seção 7.

2 CORBA

A OMG (*Object Management Group*) consiste num dos vários consórcios existentes de companhias que buscam a produção de especificações para ambientes que utilizam a orientação a objetos, voltados para sistemas distribuídos. É responsável pela definição da plataforma ORB, a qual tem por objetivo fornecer, basicamente, mecanismos pelos quais objetos fazem requisições e recebem as respectivas respostas de forma transparente, em ambientes distribuídos heterogêneos.

A CORBA obedece um modelo de objeto "clássico", onde clientes enviam mensagens para objetos e incorpora os conceitos e definições a seguir: a)**cliente**:qualquer objeto que solicita um serviço; b)**implementação de objeto**: contém o código e os dados que caracterizam o comportamento de um objeto; c)**repositório de interface**: armazena as descrições das interfaces; d)**repositório de implementações**: armazena informações que permitem a localização e ativação das implementações de objetos; e)**IDL** (*Interface Definition Language*): linguagem para descrição das interfaces; f)**ORB**: definida por suas interfaces: ***stubs*** IDL: procedimento local correspondendo a uma única operação de interfaces pré-definidas; **IID** (Interface de Invocação Dinâmica): permite a construção dinâmica de uma requisição, através de consulta ao repositório de interfaces, em interfaces conhecidas em tempo de execução; **adaptador de objetos**: provê a interface principal através da qual uma implementação de objeto efetua o acesso aos serviços da ORB; **interface ORB**: permite a interação entre cliente e implementação de objeto diretamente com a ORB, possuindo operações comuns a todas as ORBs; **esqueletos**: definidos unicamente para uma interface específica, recebe a invocação do cliente e invoca o método apropriado na implementação de objeto. A Figura 1 apresenta a estrutura geral da arquitetura.

Para requisitar um serviço, o cliente precisa possuir uma **referência de objeto** (identificador não-ambíguo de um objeto), cuja forma de implementação é uma decisão considerada fundamental, para uma ORB, no que diz respeito à estrutura, desempenho, segurança, etc. [19]. A invocação do objeto possuidor da interface desejada é realizada a partir dessa referência. O cliente não tem necessidade de saber a localização do objeto, como é efetuada a comunicação, sua implementação ou sua forma de armazenamento.

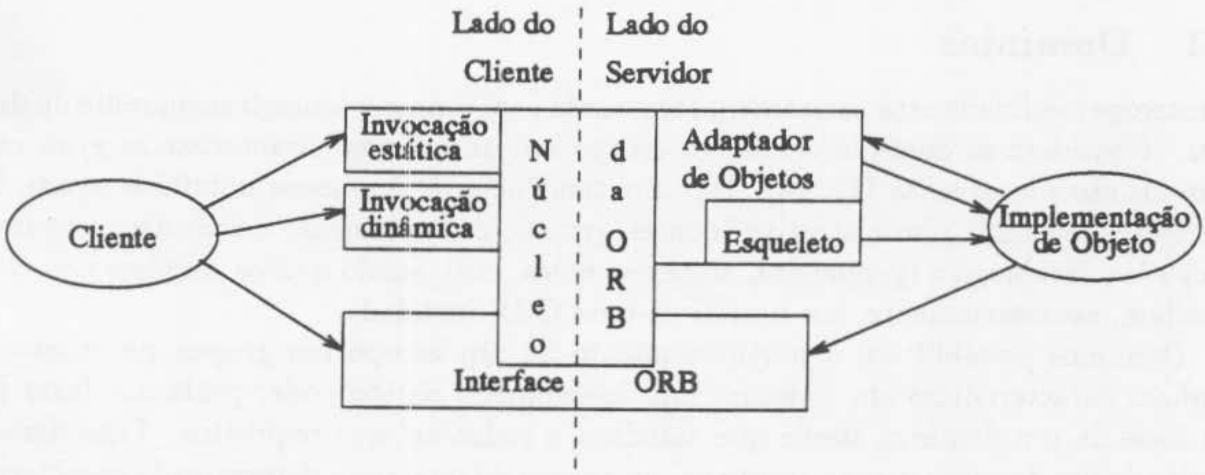

Figure 1: Estrutura da CORBA

Tendo em vista que a ORB só tem a função de possibilitar o envio e recebimento de requisições e respostas num ambiente distribuído, foram criados os Serviços de Objetos, os quais padronizam a funcionalidade básica necessária aos objetos da CORBA, tais como: persistência, ciclo de vida, transações, etc. Esses serviços podem ser agregados às ORBs, estendendo suas capacidades de forma incremental.

3 Interoperabilidade

Como previsto pela OMG, existe atualmente uma grande variedade de produtos que obedecem à especificação CORBA, entretanto, devido à grande flexibilidade permitida, as

implementações de cada ORB diferem, refletindo as soluções peculiares de cada fabricante, não só pela utilização de diferentes mecanismos para a obtenção das mesmas funcionalidades preconizadas, como também através do acréscimo de novas funções, consideradas importantes para o seu usuário final. Este fato corresponde às decisões técnicas relativas, por exemplo, ao tempo despendido por uma requisição, ao escopo abrangido por uma ORB (uma única aplicação ou todo o ambiente envolvido por uma empresa com conexões internacionais, por exemplo), aos níveis de segurança e ao uso de determinados protocolos. Essa diversidade é possível pelo fato da CORBA ter bem definidas as interfaces da ORB, não se preocupando com a forma com que serão fornecidas. A referência de objeto, por exemplo, que compõe um poderoso mecanismo nessa arquitetura, originou várias formas de implementação, tornando complexo a utilização de uma referência desenvolvida para uma ORB por outra. Outros fatores podem, também, motivar a divisão de um ambiente em ORBs diferentes, tais como: segurança, criação de ambientes para desenvolvimento e gerenciamento, tempo de vida dos objetos utilizados, sua proximidade ou não com os clientes, etc.

Dessa necessidade surge o conceito de **interoperabilidade**, que é definido em [13], como:"A habilidade de um cliente numa ORB A invocar uma operação, definida em IDL, num objeto numa ORB B, onde ORB A e ORB B são desenvolvidas independentemente".

A arquitetura descrita na especificação da CORBA 1.2 (e atualmente implementada nos produtos comercialmente disponíveis), é bastante incompleta sobre esse item, apenas dando algumas sugestões para a sua solução [4, 20]. A nova especificação, CORBA 2.0, pretende definir como a interoperabilidade pode ser alcançada.

3.1 Domínios

A interoperabilidade está basicamente associada com uma mudança transparente de **domínio**. Considera-se domínio como um escopo em que certas características e/ou regras comuns são preservadas [13, 14]. Há uma tendência de que esses domínios sejam, basicamente, de cunho administrativo (nomes, grupos, gerenciamento de recursos, segurança, etc) e/ou tecnológico (protocolos, sintaxes, redes, etc), sendo que os mesmos não correspondem, necessariamente, aos limites de uma ORB instalada.

Domínios possibilitam o particionamento de um escopo em grupos de objetos que tenham características em comum. Um determinado objeto pode, portanto, fazer parte de mais de um domínio, desde que satisfaça a todos os seus requisitos. Considera-se o limite de um domínio como o limite de um escopo no qual uma determinada característica tem algum significado. Como exemplos de domínios, podemos citar os escopos: de uma referência de objeto; de uma sintaxe de transferência de mensagens; de um endereço; de uma mensagem de rede; de uma política de segurança; de um identificador de tipos; de um serviço de transações qualquer; etc. Interoperabilidade só será possível através de uma perfeita conversão entre os domínios.

3.1.1 Interceptador

O problema básico para se conseguir interoperabilidade pode ser traduzido em como fazer para que um objeto Y, na ORB B, apareça como um objeto X, na ORB A, de maneira que esta última seja capaz de utilizar X da mesma forma que faria com um objeto qualquer que fosse, de fato, implementado por ela. Além disso, todas as funcionalidades fornecidas pela ORB B devem ser acessíveis pela ORB A através de X. Com isso, uma requisição em

X deve ser transformada numa requisição em Y e, para tanto, devemos ser capazes de criar X através da passagem de Y para a ORB A. O objeto X será, então, um representante de Y na ORB A, recebendo a denominação de *proxy* de Y [12] (Figura 2).

Durante a conversão da requisição pode ser necessário o mapeamento de outros domínios, além do definido pela referência de objeto. Múltiplos domínios podem estar sendo ultrapassados simultaneamente e cada conversão será igualmente necessária para o completo entendimento pela ORB destino. Uma forma de se conseguir interação entre duas ORBs é através da introdução de um **Interceptador**. Esse mecanismo possibilita que uma invocação iniciada numa ORB seja atendida por um objeto em outra ORB. Logicamente (computacionalmente) um Interceptador é posicionado entre os dois domínios que tem por objetivo unir (por exemplo, duas ORBs com implementações diferentes).

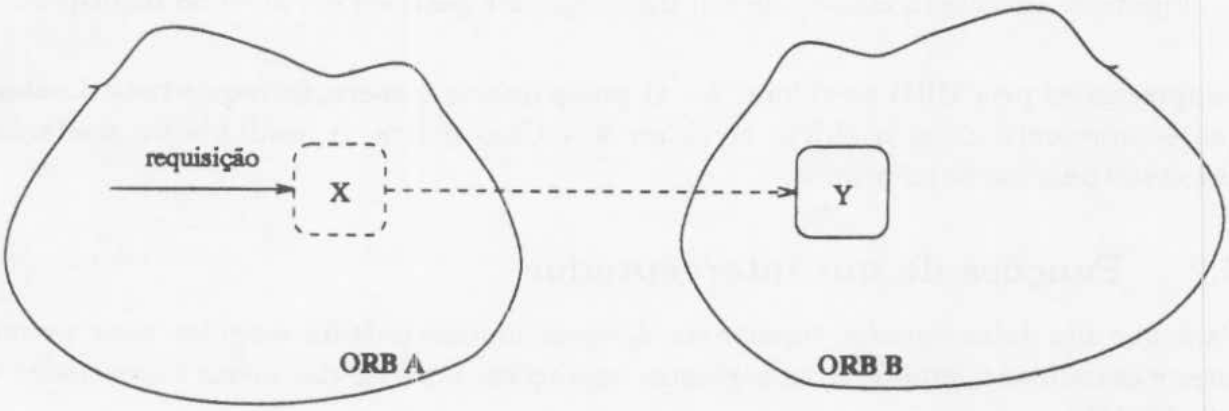

Figure 2: **X** é o *proxy* de **Y**.

O mecanismo de interceptação pode ser: a) imediato: quando as conversões são efetuadas a partir da representação de um domínio diretamente para a representação de outro domínio; ou b) intermediado: quando as conversões são efetuadas inicialmente para uma representação intermediária (canônica) e daí para a nova representação.

Pela localização de seus componentes o Interceptador pode ser: a) em linha: quando as conversões são realizadas internamente à ORB; ou b) em nível de requisição: quando as conversões são realizadas externamente à ORB. Em nível de requisição pode ser: a) específico: quando só são passíveis de conversão as interfaces predeterminadas; ou b) genérico: quando possibilita a conversão das interfaces conhecidas em tempo de execução.

Este artigo será desenvolvido sobre o mecanismo intermediado em nível de requisição e genérico, por ser mais flexível, não necessitando do conhecimento dos detalhes de implementação das ORBs envolvidas no processo de interoperabilidade, além de não se restringir às interfaces preestabelecidas. O Interceptador, então, é constituído por duas partes, uma em cada ORB, que se comunicam de forma privada (Figura 2). Também pode-se notar a invocação dinâmica que aparece na ORB A. Para que o interceptador na ORB cliente possa adquirir as informações necessárias para retransmitir a requisição para o objeto na outra ORB é necessário que se possa estabelecer dinamicamente o nome da operação e os tipos dos parâmetros. Dessa forma deve ser acrescentado um mecanismo, chamado de Interface de Esqueleto Dinâmico (IED) semelhante ao mecanismo da IID, no lado do servidor, que permita descobrir, em tempo de execução, a interface implementada [13].

O princípio de funcionamento desse tipo de interceptador obedece as seguintes etapas:

1 - A requisição original é passada para um objeto *proxy* (função do Interceptador) na ORB cliente; **2** - O objeto *proxy* converte o conteúdo da requisição para uma forma

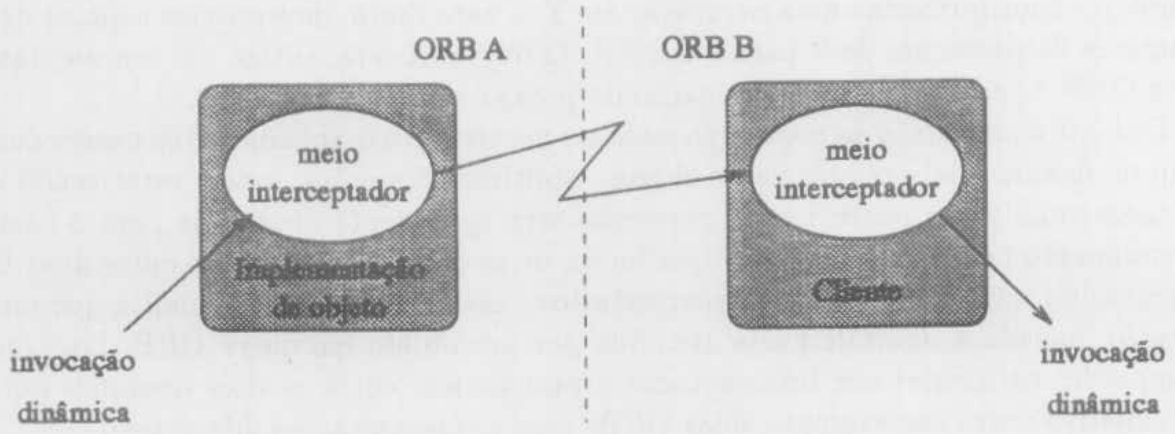

Figure 3: Invocação através de um Interceptador genérico em nível de requisição.

compreensível pela ORB servidora; **3** - O *proxy* invoca a operação requisitada no objeto que se apresenta como o objeto servidor; **4** - Caso exista, o resultado da operação é devolvido pelo caminho inverso.

3.2 Funções de um Interceptador

Para que um Interceptador suporte as diversas funcionalidades exigidas para permitir interoperabilidade, propõe-se as seguintes operações, a partir das idéias encontradas em [10, 13, 14]:

a) cria_proxy - para a criação de um ***proxy*** deve ser criado um representante desse objeto na mesma ORB em que se encontra o cliente para permitir que um objeto, instalado numa ORB B, seja acessado por um cliente, numa ORB A. Para satisfazer esse requisito, deve existir um certo objeto, em B, responsável por enviar as necessárias informações para a criação do *proxy* (o Ciclo de Vida, por exemplo, que é um Serviço de Objeto que define serviços e convenções para criar, destruir, copiar e mover objetos [9]). O Interceptador, ao receber essa informação, providencia o mapeamento da referência de objeto e aciona o Adaptador de Objetos para registrar o objeto (seu *proxy*) e gerar uma referência de objeto compatível com a ORB em que se encontra no momento.

Uma referência de objeto é associada às definições da interface e da implementação do objeto sendo referenciado, e a um dado opaco (inserido pela implementação de objeto). Durante o mapeamento da referência, não será necessário as conversões da definição de interface em virtude da especificação de um identificador único para as interfaces e a manutenção dos Repositórios de Interfaces consistentes [11]. A possibilidade da utilização da implementação do próprio Interceptador para substituir a do objeto do qual se está criando um *proxy* (sugerido em [12]), também evita a conversão da descrição da implementação. O dada opaco, por definição, não é mapeado.

De fato, o que se cria é uma referência (de objeto) para o próprio Interceptador, contendo a informação necessária que permitirá a identificação do objeto "real", cuja respectiva referência foi mapeada pelo Interceptador. Para efetuar o mapeamento, o Interceptador, que possui as propriedades de uma Implementação de Objeto, associa à referência de objeto (do *proxy*) (criada pelo Adaptador de Objetos da ORB em que está sendo criado o *proxy*) um identificador que corresponde à referência de objeto original numa tabela de mapeamento.

Uma proposta simples, considerando-se um Interceptador entre duas ORBs apenas,

exige somente a passagem da referência de objeto. A invocação de meio interceptador, na ORB A, torna claro que existe a intenção de criar um *proxy* na ORB B, dispensando a informação de ORB origem e destino. Da mesma forma, o Adaptador de Objetos não tem a necessidade de ser indicado, podendo ser sempre o mesmo (provavelmente o BOA (*Basic Object Adapter*)), já que a Implementação de Objeto será sempre o Interceptador, ou seja, o estilo da implementação não afeta a escolha do Adaptador de Objetos pela ORB que recebe o *proxy*. As necessidades características de cada implementação serão enviadas e atendidas pelo Adaptador de Objetos no qual foi registrado o objeto "real", tendo sido escolhido justamente por suportar essas exigências.

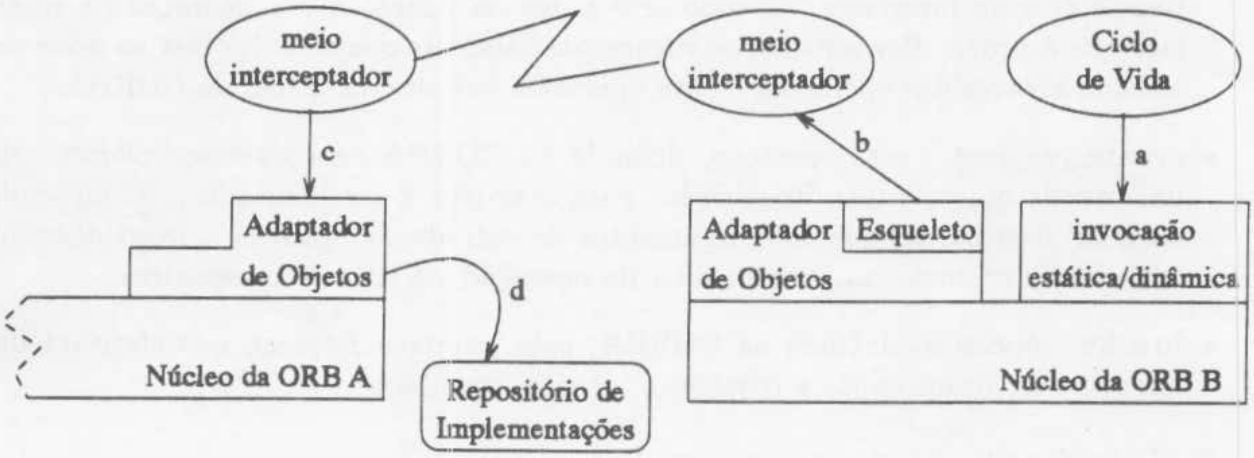

Figure 4: Criação de um *proxy*.

Na Figura 4, são identificados os seguintes passos:

1 - Um objeto qualquer (no exemplo, o Ciclo de Vida) invoca o Interceptador para criar um *proxy* de um determinado objeto na ORB A. Apesar da figura apresentar a possibilidade de invocação estática ou dinâmica, é esperado que esse tipo de invocação seja, na maioria das vezes, estática, considerando-se que a interface do Interceptador é conhecida em tempo de compilação;

2 - Um esqueleto aciona o método do Interceptador para a criação de um *proxy*. Da mesma forma que o item anterior, é esperado que a maioria das invocações seja estática;

3 - O Interceptador registra o *proxy*, através do Adaptador de Objetos; e

4 - O Adaptador de Objetos cria a referência do objeto e introduz no Repositório de Implementações a descrição da implementação (aqui representada pela descrição do Interceptador).

Para o caso do Interceptador estar servindo a mais de duas ORBs, a informação sobre a ORB_destino será necessária para permitir a identificação de qual plataforma está aceitando criar um *proxy*.

Ao receber uma invocação dirigida para um *proxy*, o Adaptador de Objetos envia ao esqueleto dinâmico (capaz de tratar qualquer tipo de interface) uma RequisiçãoDoServidor, que é o equivalente à Requisição na IED e construído a partir das informações contidas na requisição e no Repositório de Interfaces.

b) Para satisfazer as exigências da interoperabilidade, o Interceptador utilizará as seguintes operações, definidas na CORBA e propostas neste trabalho:

1 - Definidas pela interface *Object*, da qual derivam todos os objetos CORBA:

get_interface e **get_implementation** - operações definidas na CORBA que retor-

nam, respectivamente, um objeto que representa a definição de interface do objeto e um objeto que descreve a implementação do objeto em que é invocada.

2 - Definidas pela Interface de Invocação dinâmica [8]:

- **create_operation_list** - esta operação, definida na CORBA pela interface ORB, retorna uma NVList (uma lista estruturada dos parâmetros da requisição [8]), inicializada com as descrições dos argumentos para a operação a ser invocada. Recebe, como parâmetro de entrada, a definição da operação, a qual pode ser retirada do Repositório de Interfaces, sabendo-se o nome da operação e a definição da interface que a provê. Por ser uma estrutura parcialmente opaca, a NVList só pode ser alocada através das operações create_operation_list ou create_list, da CORBA;

- **create_request** - esta operação, definida na CORBA pela interface *Object*, cria um pseudo_objeto (uma Requisição) para o objeto a ser invocado . É invocada na ORB destino, tendo como argumentos de entrada os dados já convertidos pelo interceptador: contexto, identificador da operação e a lista de parâmetros;

- **invoke** - operação definida na CORBA, pela interface *Request*, que efetivamente executa a invocação após a requisição ter sido "montada".

3 - Definidas para o próprio interceptador:

- **cria_vetor** - operação proposta para prover o mapeamento entre referências de objeto. Cria um vetor com índices iguais aos **ids**, correspondendo às respectivas referências de objetos;

- **converte_ref** - operação proposta para converter uma referência de objeto de uma ORB para outra. Recebe um objeto *proxy* (ou "real"), obtém seu **id** (utilizado como índice num vetor ou como entrada numa tabela de mapeamento), faz a busca e retorna o objeto "real" (ou *proxy*) correspondente.

  Para conseguir essas informações, pode utilizar:

  - **get_id** - operação definida pela interface BOA [8], que retorna o **id** de um objeto, tendo, como parâmetro de entrada, a sua referência de objeto (no caso, do *proxy*); e

  - **busca_objref** - operação proposta que retorna a referência para o objeto do qual se possui o **id**, efetuando a busca numa tabela, por exemplo, em que está armazenado o mapeamento entre os **ids** e as suas referências de objetos.

- **converte_parâmetros** - os parâmetros são passados como uma estrutura (*NVList*) constituída por *NamedValues*, os quais deverão ter seus códigos de tipos transformados para uma forma compreensível pela nova ORB, para que seja possível, a partir daí, a construção de uma nova *NVList*. Caso haja parâmetros de retorno (inout,out) haverá também a necessidade da sua conversão para a ORB origem, o mesmo ocorrendo caso haja um resultado da operação.

- **converte_contexto** - o contexto contém informações sobre o cliente, o ambiente ou sobre a própria requisição (referentes a um Serviço de Objeto, por exemplo), consideradas inconvenientes de serem passadas como parâmetros [8];

As funções de conversão (Figura 5) poderão efetuar o mapeamento dos parâmetros da operação (incluindo o resultado) e do contexto para uma forma intermediária padronizada. Para tanto, pode ser utilizado o padrão para Representação Comum de Dados (CDR - *Common Data Representation*), apresentado em [14]. Também, de uma forma mais restrita mas potencialmente mais eficiente, pode ser efetuado o mapeamento específico desses dados diretamente entre as representações de duas ORBs particulares que pretendem interagir entre si.

4 - Definidas pelo Adaptador de Objetos:

- **cria_modificada** - Cria uma referência de objeto na ORB destino (um *proxy*), registrando-o, entretanto, com um esqueleto dinâmico. Recebe como parâmetros as definições de interface e de implementação e o dado opaco **id**.

5 - Definidas pelo objeto RequisiçãoDoServidor:

Esse objeto deve suportar operações que permitam a extração de todos os dados necessários (nome da interface e da operação, parâmetros, etc) que permitirão ao Interceptador a construção de uma requisição na ORB destino.

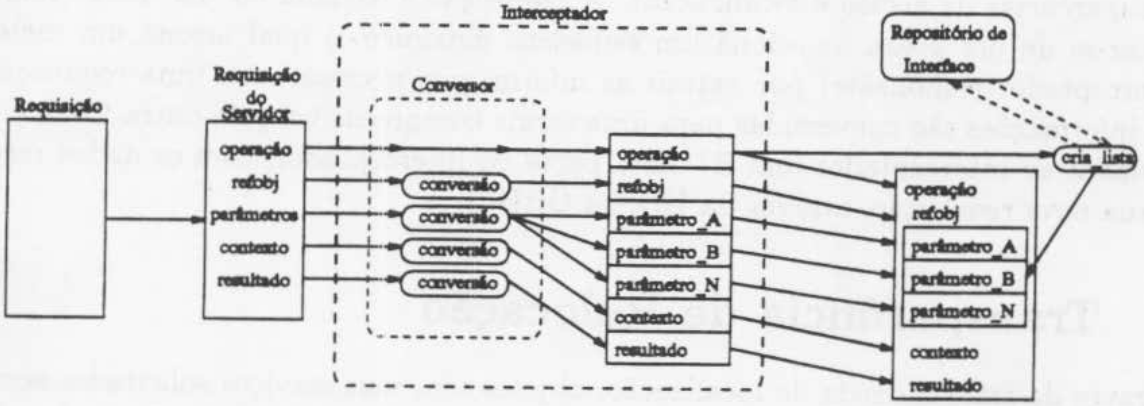

Figure 5: Mecanismo para prover interoperabilidade

3.2.1 Alterações no Adaptador de Objetos (AO)

O AO, ao receber uma invocação num objeto reconhecido como um *proxy*, direciona a requisição para uma implementação de objeto dinâmica, fornecendo uma RequisiçãoDoServidor. A operação *invoke* será a responsável por acionar todas as demais operações anteriormente descritas que colaboram com os objetivos de efetuar a transmissão de uma requisição de uma ORB para outra e de prover o mapeamento necessário de seus dados (*principal*, códigos de tipos, referências de objetos, etc). Como dado de entrada recebe uma RequisiçãoDoServidor, de onde serão extraídas todas as informações necessárias para compor uma nova requisição, através da IID na ORB destino.

3.2.2 Mapeamento de objetos entre ORBs

A especificação da CORBA, revisão 1.2, não define completamente algumas partes do sistema, visando permitir a sua especialização para diferentes aplicações e tecnologias. Dessa forma, para que duas ORBs possam cooperar, deve ser possível a conversão das referências de objetos, códigos de tipos (utilizados para descrever os parâmetros de uma interface),

principals (contém informações sobre os clientes responsáveis por uma invocação), contextos (contém informações de uma forma geral) e contextos de serviços (contém informações sobre serviços internos da ORB) [8, 14]. Os três primeiros são definidos como dados opacos; o contexto não permite o acesso a todas as informações nele contidas; não existe uma interface definida para o *principal*; e o contexto de serviço, também indefinido, passou a existir apenas na revisão 2.0 da arquitetura.

3.2.3 A forma geral do mecanismo

Um objeto X, na ORB B, pode ter seu acesso efetuado por um cliente na ORB A, através da criação de um *proxy* em A. O Interceptador (em B), usando as operações propostas, recebe sua referência de objeto, cria um identificador próprio para essa referência e envia o identificador ao Interceptador (em A), o qual registra esse objeto, associando o identificador à referência de objeto obtida durante o seu registro. De posse de uma referência desse tipo, o cliente pode efetuar uma invocação normalmente (garantindo, portanto, as transparências de acesso e localização). A requisição é enviada ao AO. Este, detectando tratar-se de um *proxy*, seleciona um esqueleto dinâmico, o qual aciona um método no Interceptador responsável por extrair as informações necessárias a uma requisição. Essas informações são convertidas para uma forma compreensível pela outra ORB e, então, enviadas ao Interceptador (em B). Esta parte do Interceptador, com os dados recebidos, efetua nova requisição, através da IID, na ORB B.

4 Transparência de Relocação

Através da transparência de localização, objetos têm seus serviços solicitados sem que o cliente necessite possuir qualquer indicação de sua posição. A transparência de relocação estende esse conceito para objetos que se movimentaram entre invocações [7]. A ORB garante a transparência de localização e acesso. Vamos apresentar, então, um mecanismo que permita transparência de relocação, a qual não é atualmente contemplada pela especificação CORBA e depois a estenderemos entre ORBs. Ocorrendo a movimentação de um objeto, torna-se necessário um mecanismo que possibilite que as novas requisições sejam enviadas ao objeto na sua nova localização, de forma transparente ao cliente.

4.1 Relocação no escopo de uma mesma ORB

Nesse escopo, introduzimos um objeto, ao qual chamamos de **Relocador**, responsável por estabelecer o mapeamento entre os identificadores dos objetos provedores de serviços e a localização para onde foram movimentados. Esse objeto obedece aos conceitos básicos definidos no RM-ODP [3].

O objeto responsável pela relocação, provavelmente um Serviço de Objeto (Ciclo de Vida, Externalização, Instalação e Ativação, ou qualquer outro objeto que venha a ser desenvolvido com essa funcionalidade[1]), também será o responsável por atualizar as informações do Relocador.

[1] A *Object Service Architecture* (OSA) 8.1 [15] sugere a criação de um Serviço de Objeto específico para relocação.

O Relocador gerencia uma tabela na qual são armazenados os identificadores dos objetos movimentados (sua referência de objeto ou nome, por exemplo) e suas correspondentes novas localizações. O Relocador proposto possui as seguintes operações:

- **localiza** - recebe como parâmetro de entrada um identificador correspondente ao objeto que efetuou, ou está efetuando, uma movimentação e retorna um dos seguintes parâmetros:

  - a nova localização do objeto movimentado, no formato utilizado pela plataforma para localizar um objeto: seu novo endereço IP e o *path* do arquivo (executável ou de dados), por exemplo;
  - uma mensagem informando que o referido objeto ainda se encontra em movimentação, caso o Relocador, tendo sido comunicado que o objeto estava iniciando uma relocação, ainda não recebeu a atualização de sua localização; ou
  - um valor nulo, caso o objeto não esteja armazenado no Relocador, caso em que ocorreu uma falha.

- **insere_inicio_movimentacao** - recebe como parâmetro de entrada o identificador do objeto que iniciou a movimentação. Inicialmente, chama a operação **localiza** para verificar se já não existe uma nova localização correspondente a esse objeto. Essa situação pode ocorrer devido a sucessivas movimentações no escopo de uma mesma ORB, portanto, apenas a localização existente deve ser substituída por um sinal de início de movimentação. Tendo a operação **localiza** retornado um valor nulo, significa que pode ser inserido o objeto, juntamente com o sinal de início de movimentação;
- **insere_nova_localizacao** - recebe como parâmetros de entrada um identificador do objeto movimentado e a sua localização atual e, daí, executa uma busca usando esse identificador para introduzir a nova localização do referido objeto.
- **remove** - recebe como parâmetro de entrada um identificador do objeto movimentado, retirando-o do Relocador juntamente com suas informações correspondentes. Esta função pode ser utilizada, por exemplo, em decorrência da atualização do Repositório de Implementações, ou das informações contidas em qualquer que seja o dispositivo utilizado pela ORB para efetuar a localização dos objetos (Serviço de Nomes, *Trading*, etc). Deve ser efetuada obedecendo uma política coerente com as necessidades que originaram, inicialmente, a relocação dos objetos.

Para que um objeto possa ser movimentado, ele precisa "colaborar". Nesse caso, significa que deve herdar a interface do Ciclo de Vida [15]. Para suportar a transparência de relocação, propomos estender a funcionalidade desse objeto, o qual será o responsável por acionar o Relocador. Sua interface provê uma operação que move um objeto para o escopo de um **localizador de fábricas**. Fábrica é definida como um objeto com a capacidade de criar outro objeto, fornecendo ao cliente as operações necessárias para criar e inicializar novas instâncias. Para criar um objeto, um cliente deve, necessariamente, possuir uma referência de objeto para uma fábrica que atenda as características do objeto em movimentação. Fábricas não são objetos especiais, possuindo as mesmas características de um objeto qualquer em CORBA. Tanto a implementação quanto a definição das interfaces das fábricas são consideradas parte do desenvolvimento da aplicação [9].

As seguinte operações devem ser acionadas no Relocador, através do Ciclo de Vida:

- **insere_inicio_movimentacao** - usando como parâmetro de entrada o identificador do objeto a ser movimentado;

- **insere_nova_localizacao** - a nova localização pode ser extraída da localização da fábrica que será utilizada na movimentação do objeto.

4.2 Relocação entre ORBs com diferentes implementações

Para suportar a transparência de relocação, são necessárias as seguintes extensões:

a) Objeto que efetua a relocação - deve enviar, junto com as informações necessárias à relocação, uma **chave** que identifique, de forma única e confiável, o objeto sendo movimentado (seta **a**, na Figura 6). Tal identificador não necessita ser interpretado pela ORB destino, sendo compreendida apenas pela ORB que a originou, portanto, pode ser transmitida como um dado opaco. A fim de garantir a integridade de tipos, a chave deve conter o identificador global da interface [11] e, caso haja mais de uma ORB, pode ser acrescentado um identificador da ORB (seu nome ou um identificador qualquer registrado com a OMG, por exemplo). Apesar de omitida na Figura 6, essa informação também deve ser interceptada e as referências para as fábricas já devem estar disponíveis para a ORB A, no caso. Essa chave é enviada ao Interceptador (seta **b**, na Figura 6), onde será armazenada para futura verificação, durante o mapeamento dos objetos. Essa chave também é introduzida no Relocador (seta **c**, na Figura 6), onde será interpretada como um sinal representando que o objeto ao qual corresponde está em processo de movimentação.

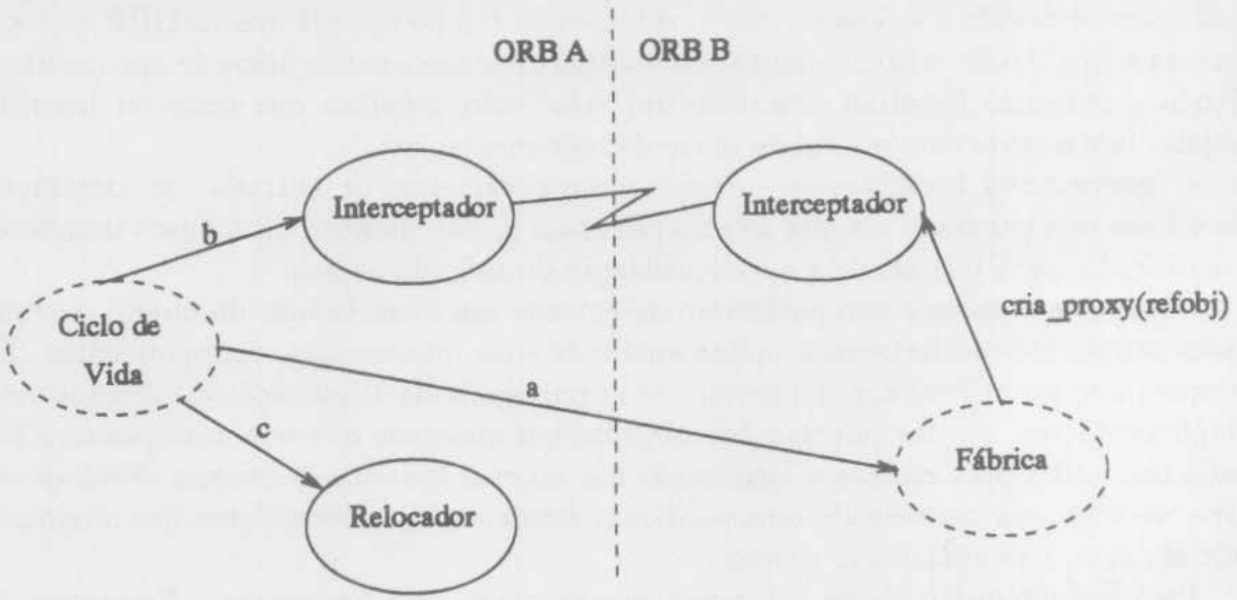

Figure 6: Mecanismo para suporte à transparência de relocação

Ao ser movimentado, pode-se considerar que o objeto está sendo instalado na nova ORB, portanto, deve ser registrado, através do Adaptador de Objetos, a fim de tornar-se disponível para utilização pelo sistema e gerar uma nova referência de objeto, compatível com a nova plataforma. É responsabilidade do objeto que efetuou a relocação providenciar que o objeto migrado torne-se disponível para a sua ORB de origem. Para tanto, o Objeto que efetua a relocação deve ser capaz de prover meios para que seja efetuada uma invocação ao **Interceptador**, a partir da ORB destino da movimentação, de forma a possibilitar o mapeamento entre as referências de objetos das plataformas em questão. Juntamente com a referência é necessário que seja enviada a chave.

Usando as definições para o Ciclo de Vida, podemos agregar essa funcionalidade ao objeto **fábrica**, o qual tem a função de criar o objeto sendo movimentado na sua nova

ORB. A referência de objeto decorrente dessa criação, juntamente com a chave, é enviada ao Interceptador (através da operação cria_proxy). A chave pode ser inserida nas informações dos dados da referência de objeto.

b) **Interceptador** - ao ser invocado para prover o mapeamento entre as referências de objetos utilizadas por uma ORB e outra, o Interceptador verifica se existe uma chave agregada à referência a ser mapeada, o que indica que o objeto que está sendo apresentado, na verdade, pertenceu, em algum momento, à própria ORB para a qual está tendo sua referência convertida. Percebendo esse fato, o Interceptador pode:

1 - introduzir no Relocador, junto à referência de objeto indicada pela chave, a própria localização do Interceptador, acrescida de um sinal informando tratar-se de um *proxy*. Pode ser efetuado desta forma, uma vez que o Interceptador funciona como um representante local do objeto remoto, para o qual repassa as mensagens referentes a uma invocação e do qual recebe as respostas, caso existam, entregando-as ao cliente (seção 3.2). A operação **insere_nova_refobj** é a responsável por essa inserção.

O esquema de envio da requisição, portanto, permanece o mesmo, sendo que a invocação é desviada para o respectivo Interceptador e, ao chegar no Adaptador de Objetos, recebe, ainda na ORB cliente, o tratamento de um *proxy*.

2 - usar a referência de objeto contida no Relocador para efetuar a troca da definição de implementação de objeto indicada por essa referência pela definição do próprio Interceptador. Isto pode ser efetuado pela operação do Adaptador de Objetos, definida pela CORBA, ***change_implementation***, que pode ser chamada normalmente pelo Interceptador. Este procedimento é mais genérico uma vez que pode usar o mesmo mecanismo de resolução de referência adotado pela ORB em utilização. Outra diferença é que a funcionalidade da operação cria_proxy, ao detectar um objeto relocado pela presença da chave, não mais cria um novo objeto (que seria um *proxy*), mas altera a definição de implementação associada à referência de objeto. O Interceptador deve, então, avisar ao Relocador que sua referência de objeto continua válida. Dessa forma, é necessário uma nova resolução de referência para efetuar nova requisição. Da mesma forma, a requisição será enviada ao Interceptador.

c) **Relocador** - além das funções já descritas, deve passar a suportar as seguintes operações:

- **insere_nova_refobj** - usada para a inserção dos identificadores dos objetos. É acionada pelo Interceptador, que é o objeto que "conhece" quais os objetos que devem ter suas novas localizações inseridas no Relocador.
- **substitui_refobj** - faz a substituição da referência de objeto que estava sendo utilizada para a invocação por outra, fornecida pelo Relocador, representando um *proxy* do objeto movimentado. Deverá poder acessar a requisição que está sendo enviada, a fim de executar essa alteração e iniciar a "montagem" de uma **Requisição do Servidor** [13].

d) **Objeto Relacionamento** - a existência de um objeto que referencie outros é uma situação comum num ambiente distribuído. Portanto, quando se pensa em movimentação de objetos para outras ORBs, surge a preocupação com as conseqüências desse fato, pois um fornecedor de serviços pode utilizar serviços de outros objetos, os quais permanecem na ORB origem e, portanto, inacessíveis ao objeto agora deslocado para outra ORB.

Esse objeto gerencia um repositório que contém a referência de objeto de todos os objetos de uma ORB bem como uma relação de todos os outros objetos que cada um referencia. Deve suportar as seguintes operações:

- **insere_objeto** - insere objeto com respectivos objetos referenciados;
- **insere_ref** - insere novas referências para um objeto já existente (para o caso de ter havido alguma alteração no objeto);
- **remove_objeto** - retira o objeto e suas referências da tabela;
- **busca_ref** - busca o objeto e retorna todas as suas referências; e
- **remove_ref** - retira apenas as referências determinadas do objeto refobj. Para o caso de algumas das referências utilizadas pelo objeto deixarem de existir, o que pode acontecer quando o objeto não utiliza todos os objetos referenciados e alguns deles têm sua funcionalidade agregada a outros.

Quando um determinado objeto for movimentado o Relocador verifica, no objeto Relacionamento, quais são as suas referências e, então, as envia ao Interceptador para que este crie um *proxy* para cada um dos objetos referenciados (através da operação **criar_proxy**). Com tal procedimento, esses objetos poderão ser invocados pelo objeto movimentado como se estivessem todos fazendo parte de uma mesma ORB.

5 Descrição da implementação

Para o desenvolvimento de nosso trabalho sobre interoperabilidade utilizou-se a ORBeline [18], da *PostModern*, usando o compilador C++, da Sun, e um protótipo de ORB descrito em [1], desenvolvido em C++,da *Free Software Foundation*, e o RPC, da Sun. É conveniente ressaltar que muitas das implementações atualmente existentes de ORB baseiam-se em RPC [4, 16, 22].

O protótipo utilizado apresenta a funcionalidade básica mínima da CORBA, não tendo sido implementados a Interface de Invocação Dinâmica, o Repositório de Interfaces, e alguns recursos da IDL, como exceções e contextos. Sua estrutura básica aparece na Figura 7, em linha contínua. Os objetos tracejados foram acrescentados para suportar a interoperabilidade e a transparência de relocação. Note que o Interceptador é uma Implementação de Objeto.

O ***portmapper*** tem a finalidade de identificar a porta na qual o servidor está escutando, recebendo como parâmetros a localização do servidor e o identificador da classe. O **Repositório de Implementações** é usado para localizar um servidor, recebendo como parâmetro a classe dos objetos por ele implementados. É implementado como um arquivo contendo um registro para cada servidor existente na rede, composto pelo identificador da classe que implementa e o endereço IP da máquina em que se encontra. A classe, neste sistema, consiste na interface do serviço a ser solicitado. O **Repositório de Objetos** consiste numa lista de referências cruzadas entre as referências de objetos e seus ponteiros. O Adaptador de Objetos é acionado pela implementação para gerar uma referência para aquele objeto. Ao receber uma solicitação de serviço, o adaptador de objetos busca a referência de objeto utilizada na requisição e, utilizando o ponteiro correspondente, efetua a chamada ao método adequado. Os ***stubs*** da ORB têm a finalidade de repassar a invocação e seus parâmetros para o sistema RPC. O *stub* RPC prepara os parâmetros para serem utilizados pela rotina de suporte do RPC responsável pela invocação, a qual efetua a conversão dos parâmetros, através das bibliotecas XDR (*eXternal Data Representation*), e os transmite. Processo inverso ocorre com a resposta, sendo convertida do formato XDR para uma forma compreensível pelo *stub* RPC e, então, enviada ao *stub* ORB. Processo semelhante ao da resposta, ocorre no lado do servidor. Os parâmetros são convertidos

do formato XDR para o formato compreensível pelo *stub* RPC, o qual aciona o esqueleto ORB correspondente ao método desejado.

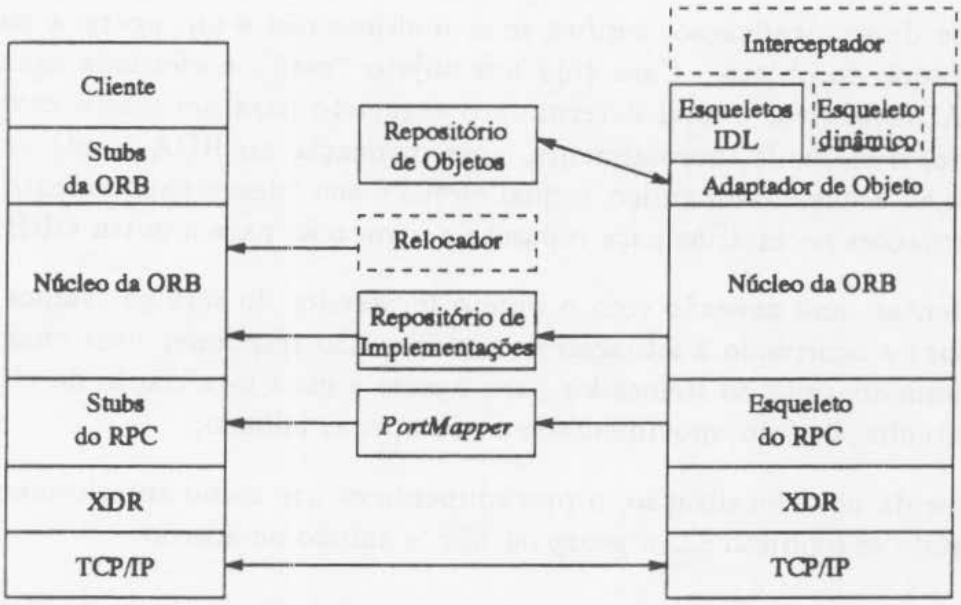

Figure 7: Arquitetura do sistema, apresentando o Relocador e o Interceptador (parte do cliente)

Cada método, da classe que está sendo implementada, será associado a um *stub* e um esqueleto, gerados em linguagem C++, os quais deverão ser integrados aos processos do cliente e do servidor, respectivamente. A referência de objeto foi implementada como uma estrutura contendo o endereço IP da máquina, um identificador do serviço e um identificador do objeto dentro da classe a qual implementa.

A ORBeline, versão 1.0, implementa a CORBA revisão 1.1, com todas as suas funcionalidades.

Nossa implementação possibilita, com algumas restrições:

- a transparência de relocação no protótipo de [1];
- a invocação, de forma transparente, a partir do protótipo, de serviços que tenham sido registrados na ORBeline e colocados disponíveis ao protótipo; e
- a transparência de relocação de um objeto que tenha se movimentado do protótipo para a ORBeline.
- a extensão das transparências de acesso e localização para os serviços fornecidos por objetos na ORBeline e tornados disponíveis ao protótipo.

5.1 Implementação da transparência de relocação

O Relocador foi implementado como um objeto que tem acesso a um arquivo contendo: a referência de objeto (ou seja, o nome da máquina na qual o objeto estava originalmente, o identificador da interface do objeto, o identificador do objeto) e o nome da máquina para a qual o objeto movimentou-se. Quando se faz necessário uma consulta, basta efetuar a verificação pelo nome da máquina origem da movimentação.

Todas as requisições são enviadas ao Adaptador de Objetos, que foi implementado juntamente com os esqueletos e o Interceptador, num só processo.

São obedecidos os seguintes passos, no decorrer de uma invocação:

- Inicialmente, busca-se, no repositório de implementações, a localização de um servidor que forneça a interface que se deseja;
- De posse dessa localização, verifica-se se o objeto não é um *proxy*, a partir do seu identificador de objeto. Caso seja um objeto "real", é efetuada uma invocação ao BOA respectivo, o qual determina o esqueleto para onde será enviada. Caso contrário, é efetuada, normalmente, uma invocação ao BOA e este entrega a requisição ao esqueleto dinâmico, o qual efetua o seu "desempacotamento" extraindo as informações necessárias para repassar a invocação para a outra ORB;
- Ao se tentar uma conexão com o objeto fornecedor do serviço (vamos chamar de **servidor**) e ocorrendo a situação do servidor não responder uma chamada, é efetuada uma consulta ao Relocador para buscar a nova localização do objeto, caso o mesmo tenha, de fato, movimentado e não apenas falhado;
- De posse da nova localização, o procedimento ocorre como anteriormente descrito, verificando se o objeto é um *proxy* ou não, e agindo de acordo.

A movimentação do objeto, fora do escopo desse trabalho, é realizada através da destruição do processo que fornece o serviço solicitado, na localização em que está sendo invocado, seguido de sua instanciação em outra localização. Durante o processo de instanciação, a própria implementação de objeto efetuará o registro do objeto na nova ORB, através do Adaptador de Objeto, e no Interceptador, tornando-o disponível para acesso pela ORB de origem. As funções do Relocador foram implementadas de acordo com a seção 4.

5.2 Implementação do Interceptador

O Interceptador é composto por duas partes: uma como Implementação de Objeto no protótipo de [1] (ORB A, na Figura 3), ao qual chamamos Interceptador-servidor e outra como um cliente na ORBeline (ORB B, na Figura 3), o Interceptador-cliente. A transmissão das informações entre esses meio-interceptadores pode ser realizada por várias formas [12] entre as quais, podemos citar: *sockets*, TLI, RPC ou mesmo uma terceira ORB. Para a nossa implementação escolheu-se *sockets*. Foram implementadas as operações definidas para o Interceptador na seção 3.2, exceto as de conversão de parâmetros e de contexto.

5.2.1 Comportamento do Interceptador na ORB cliente

O Interceptador é implementado juntamente com o Adaptador de Objetos e o esqueleto dinâmico. Ao receber uma invocação, o Adaptador detecta tratar-se de um *proxy*, pela chave, e aciona o esqueleto dinâmico, que irá extrair as informações da requisição que chega e fornecê-las ao Interceptador. Como o protótipo base utilizado não implementa o Repositório de Interfaces nem a IID, a requisição foi implementada como uma estrutura em que são enviados os códigos dos tipos de cada parâmetro, e é dessa estrutura que são retirados os argumentos a serem convertidos e enviados para o Interceptador da ORB B. Da mesma forma, só foi efetivamente implementada a conversão para referências de objetos.

5.2.2 Comportamento do Interceptador na ORB servidora

O Interceptador recebe os dados enviados pelo Interceptador na ORB A e efetua sua conversão para a forma utilizada pela ORBeline. Esse Interceptador, como qualquer cliente da ORB, deverá consultar o Repositório de Interfaces para extrair os dados necessários para efetuar uma invocação dinamicamente.

Inicialmente, o Interceptador-cliente consulta uma tabela contendo a chave recebida do Interceptador-servidor e a respectiva referência de objeto.

Portanto, o Interceptador-cliente será o responsável pela conversão da cadeia de caracteres para uma referência de objeto (através da operação *_string_to_object*, da interface ORB). Essa operação retorna um objeto da classe base em Object que servirá como parâmetro de entrada para a operação de criação da requisição dinâmica. Para essa criação usaremos como parâmetros a referência de objeto, o nome da operação e o nome do Repositório de Interfaces, onde se encontra a interface. A operação de criação dinâmica, na ORBeline, facilita o trabalho dos programadores das aplicações por efetuar automaticamente a consulta ao Repositório de Interfaces, evitando a necessidade da utilização de um navegador, por exemplo, para extrair as informações daquele repositório.

A referência de objeto é utilizada para a obtenção do nome da interface, a qual será usada como parâmetro de busca no Repositório de Interfaces. Além disso, essa operação já constrói uma espécie de "gabarito", com seus campos inicializados: o nome da interface, a identificação da operação, o modo (in, out, inout) e o código de tipo de cada parâmetro. A partir daí, o programador deve "preencher" os valores correspondentes aos parâmetros (no caso, só os que possuem modo in ou inout).

Os valores dos parâmetros recebidos, após serem convertidos para o formato da ORBeline, são introduzidos no gabarito. Entretanto, para que isso seja possível, todos os parâmetros devem ser inicializados numa instância de uma classe, específica para cada tipo, derivada da classe base *Value*. Essa classe consiste numa forma genérica para representação de dados. Para cada tipo de dados, existe uma classe que deriva de *Value*, possuindo operações para acesso aos dados. Através dos códigos de tipos são construídas instâncias apropriadas. Através de uma fábrica, que recebe como parâmetro um código de tipo (por referência ou por ponteiro) e retorna um ponteiro para um objeto da classe *Value*, para o tipo especificado, já alocando a quantidade necessária de memória.

De posse do código de tipo, retirado do Repositório de Interfaces, podemos identificar o tipo do parâmetro e, então, usar o valor recebido do Interceptador-cliente e convertido para o formato da ORBeline, para inicializar uma instância da classe *Value*. Encapsulado dessa forma, o valor recebido pode ser introduzido na requisição. Por exemplo, para os tipos primitivos:

```
...
CORBA_DII::REQUEST* REQ = CORBA_DII::REQUEST::CREATE(/* ARGUMENTOS */);
CORBA::TYPECODE::TCKIND TIPO = CODIGO_TIPO.KIND();
CORBA::VALUE *VALOR = REQ- >ARG("NOME_PARAMETRO");
VALOR->ULONGVALUE(VALOR_ULONG);
...
```

A partir de uma requisição (req), obtém-se um ponteiro para uma classe, derivada de *Value*, construída para o tipo determinado pelo parâmetro "nome_parametro". Para cada tipo primitivo existe uma operação para inserir o seu valor nas suas respectivas classes. No exemplo, por tratar-se de um inteiro longo sem sinal (representado por *ulong*), utiliza-se a

operação correspondente ulongValue(), que deve receber um valor de entrada compatível.

Também seria possível esse resultado pela obtenção do código de tipo do parâmetro "nome_parametro" (através da operação arg_type), a ser utilizado na construção de um *Value* para o tipo especificado (operação factory(CORBA::TypeCode& codigo_tipo)). A operação de acesso a ser utilizada pode ser descoberta através da operação kind().

Depois de "montada", a requisição pode ser enviada ao objeto fornecedor do serviço, observando-se todo o procedimento normal de uma invocação dinâmica.

6 Trabalhos correlatos

Atualmente, pode-se notar um esforço muito grande por parte da OMG para definir completamente as soluções encontradas para permitir a interoperabilidade entre ORBs. Este esforço resultou na especificação UNO (*Universal Networked Objects*) [14], a qual apresenta o GIOP (*General Inter-ORB Protocol*), que suporta interoperabilidade em nível de protocolo especificando uma sintaxe de transferência comum (CDR) e padronizando o formato das mensagens trocadas entre ORBs. Como uma especialização dessa especificação surgiu o IIOP *Internet IOP*, que consiste num GIOP sobre TCP/IP. Também foram criados os ESIOPs (*Environment Specific IOP*), que são protocolos específicos para um determinado ambiente. Até o momento, só faz parte da especificação o ESIOP da DCE (*Distributed Computing Environment*). A UNO também preconiza a necessidade do suporte ao mecanismo de interceptação inter-ORBs. Está prevista uma revisão da CORBA 2.0 para janeiro/96.

O trabalho apresentado em [21] descreve uma estrutura para a construção de um mecanismo de interceptação, obedecendo aos conceitos definidos pela OMG e provendo interoperabilidade entre duas implementações de ORB (Orbix, da IONA Technologies, e DOME, da Object -Oriented Technologies).

7 Conclusão

Neste trabalho apresentou-se a descrição de um mecanismo para permitir a interoperabilidade entre ORBs implementadas com tecnologias diferentes e uma forma de prover suporte à transparência de relocação, estendida para o caso de ORBs interoperantes. Também foi descrito uma forma de implementação desses mecanismos, utilizando-se uma plataforma comercial e um protótipo construído sobre RPC. Deste trabalho conclui-se:

- O mapeamento das requisições e seus parâmetros entre ORBs, ou de uma ORB para o formato intermediário, é totalmente dependente da implementação das ORBs. Este mapeamento pode ser realizado por um módulo separado do Interceptador [21]. As conversões do contexto e do *principal* (um identificador do cliente), apesar de serem simples, exige um acordo de alto nível (administrativo, por exemplo) para garantir sua semântica;

- O mapeamento das referências de objetos é fundamental para a interoperabilidade e pode ser obtido através de um mecanismo relativamente simples;

- Sem a consistência dos Repositórios de Interfaces das ORBs o problema ficaria muito mais complexo, obrigando o mapeamento e a passagem das descrições das interfaces entre as ORBs;

- A existência de um *browser* "embutido" na IID da ORBeline e a classe *Value* facilitou bastante a programação do Interceptador.

Observa-se que, em virtude da grande flexibilidade fornecida por um Interceptador genérico, sua complexidade é bastante aumentada, resultando numa queda de desempenho razoável, uma vez que cada invocação resultará em várias consultas (e invocações) efetuadas pela ORB e pelo Interceptador, semelhante ao que acontece com a IID [22, 20]. Essa desvantagem é bastante reduzida com a utilização de um Interceptador específico, embora os serviços a serem solicitados fiquem restritos aos previamente estabelecidos. É necessário uma análise de qual seria a opção mais vantajosa observando-se as necessidades atuais e futuras das ORBs em questão. Em vista disso, uma extensão de nosso trabalho seria a descrição de um Interceptador originalmente específico mas que permitisse ser estendido, em virtude da evolução dos objetivos das ORBs as quais comunica, para tratar interfaces desconhecidas até então. Esta extensão deveria ter flexibilidade suficiente para escolher (em relação à Figura 3) uma invocação dinâmica ou estática na ORB A e, independente dessa decisão, uma invocação dinâmica ou estática na ORB B.

Agradecimentos

Agradecemos à FAPESP e ao CNPq, pelo apoio financeiro, e á Marinha do Brasil, pelo tempo concedido.

References

[1] A. M. V. de Mello. Um Protótipo de Negociador de Requisições de Objetos. Master's thesis, UNICAMP - FEE, 1995.

[2] E.R.M.Madeira. Multiware Platform: Some Issues about the Middleware Layer. In *7th IASTED International Conference on Parallel and Distributed Computing and Systems*, Washington, Estados Unidos, pages 162–166, outubro 1995.

[3] ITU-T — ISO/IEC. *Draft Recommendation X.902: Basic Reference Model of ODP - Part 3*, 1995.

[4] J.R.Nicol, C.T.Wilkes, and F.A.Manola. Object Orientation in Heterogeneous Distributed Computing Systems. *IEEE Computer*, pages 57–67, Junho 1993.

[5] M.Betz. Interoperable objects. *Dr. Dobb's Journal*, 19(11), outubro 1994.

[6] M.J.Mendes and E.R.M.Madeira. Plataforma Multiware: Projeto e Desenvolvimento da Camada Middleware. In *12o. Simpósio Brasileiro de Redes de Computadores - SBRC 94*, Curitiba - PR, pages 93–109, maio 1994.

[7] N.Davies, G.S. Blair, and J.A.Mariani. Supporting persistent re-locateble objects in the ANSA architecture. Technical report, Computing Department, Lancaster University, 1992. MPG-92-04.

[8] OMG. *Common Object Request Broker: Architecture and Specification*, Dezembro 1994. OMG 93-12-29.

[9] OMG. *Common Object Services Specification,vol. 1*, março 1994. OMG 94-1-3.

[10] OMG. *OMG ORB2.0 Interoperability and Initialisation RFP Response*, março 1994. BNR Submission - OMG TC Document 94.3.4.

[11] OMG. *OMG RFP Submission - Interface Repository*, novembro 1994. OMG TC Document 94.11.7.

[12] OMG. *ORB 2.0 RFP Submission - ORB Interoperability*, março 1994. IONA/Sun Submission - OMG TC Document 94.3.1.

[13] OMG. *ORB 2.0 RFP Submission - Universal Networked Objects*, setembro 1994. OMG TC Document 94.9.32.

[14] OMG. *CORBA2.0/ Interoperability - Universal Networked Objects*, março 1995. OMG TC Document 95.3.xx[REVISED 1.8jm].

[15] OMG. *Object Services Architecture*, Janeiro 1995. OMG Document 95.1.47 rev. 8.1.

[16] R. Orfali and D. Harkey. Client/Server with Distributed Objects. *Byte*, pages 151–162, abril 1995.

[17] R. Orfali, D. Harkey, and J. Edwards. Intergalactic Client/Server Computing. *Byte*, pages 108–122, abril 1995.

[18] PostModern Computing Technologies, Inc. *ORBeline User's Guide*, 1994.

[19] M. L. Powell. *Objects, References, Identifiers and Equality - White Paper.* OMG, julho 1993. OMG TC Document 93-7-5.

[20] D. C. Schmidt and S. Vinoski. Object Interconnections - Comparing Alternative Client-side Distributed Programming Techniques. *C++ Report*, 7(4), maio 1995.

[21] M. Steinder, A. Uszok, and G.K. Zieliński. *Distributed Platforms - A Framework for Inter-ORB Request Level Bridge Construction.* Chapman & Hall, pp., 1 edition, fevereiro 1996.

[22] S. Vinoski. Distributed Object Computing With CORBA. *C++ Report*, Julho/Agosto 1993.

[23] W.P.C.Loyolla, E.R.M.Madeira, M.J.Mendes, E.Cardozo, and M.F.Magalhães. Multiware Platform: An Open Distributed Environment for Multimedia Cooperative Applications. In *IEEE Computer Software & Applications Conference - COMPSAC 94*, Taipei, Taiwan, novembro 1994.