



UFPA

Universidade Federal do Pará

*INTEROPERABILIDADE EM SISTEMAS
HETEROGENEOS REAIS: IMPLEMENTAÇÃO E
AVALIAÇÃO POR TÉCNICA DE AFERIÇÃO DE DADOS*

Adriana de Nazaré Farias da Rosa

2. semestre / 2006

Centro Tecnológico
Universidade Federal do Pará
Campus Universitário do Guamá
Belém - Pará

Universidade Federal do Pará
Centro Tecnológico
Engenharia da Computação

Adriana de Nazaré Farias da Rosa

*INTEROPERABILIDADE EM SISTEMAS
HETEROGENEOS REAIS: IMPLEMENTAÇÃO E
AVALIAÇÃO POR TÉCNICA DE AFERIÇÃO DE DADOS*

Trabalho submetido ao Colegiado do Curso de
Engenharia da Computação para obtenção do
grau de Engenheira de Computação

Orientador: Prof. Dr. Carlos Renato Lisboa Francês
Doutor em Ciências da Computação e Matemática Computacional - ICMC - USP

Co-orientador: Msc. Edvar da Luz Oliveira
Mestre em Engenharia Elétrica - UFPA

Belém - PA

2006

*INTEROPERABILIDADE EM SISTEMAS
HETEROGENEOS REAIS: IMPLEMENTAÇÃO E
AVALIAÇÃO POR TÉCNICA DE AFERIÇÃO DE DADOS*

Este trabalho foi julgado em ___ / ___ / _____ adequado para obtenção do Grau de Engenharia de Computação, e aprovado na sua forma final pela banca examinadora que atribuiu o conceito _____.

Prof. Dr. Carlos Renato Lisboa Francês
ORIENTADOR - UFPA

Msc. Edvar da Luz Oliveira
CO-ORIENTADOR - UFPA

Prof. Dr. João Crisóstomo Weyl A. Costa
MEMBRO DA BANCA EXAMINADORA - UFPA

Prof. Dr. Manoel Ribeiro Filho
COORDENADOR DO CURSO - UFPA

Belém - PA

2006

Aos meus pais e irmã.

Aos amigos, pelo apoio e companheirismo.

Agradecimentos

A Deus por ser a Luz em meu caminho.

A minha família, pelo apoio incondicional em todos os momentos difíceis, pois foram, são e sempre serão o alicerce de minha vida.

Ao professores e amigos, João Crisóstomo, Renato Francês e Edvar Oliveira, pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Ao meu Éder, carinhosamente “meu vida”, por estar sempre ao meu lado, segurando minha mão quando tudo se tornava difícil, por sofrer, chorar, sorrir, e caminhar comigo... Não só durante esta “reta final”, mas sim, durante toda nossa primeira jornada de estudos. A você, com todo meu carinho, Meu sincero e singelo - MUITO OBRIGADA!

Aos meus amigos do peito - Tias: Cyn, Ni, Livi e Lili; Iguetchhhhh, Markinhos e Markinhos_Style. Por todos os momentos de descontração, apoio, garra e amizade. Por provarem que a amizade é bem mais que um sentimento, ela se traduz em atos e palavras, e se eterniza como uma sólida rocha para todo o sempre.

A família LEA, pela calorosa acolhida, apoio, e momentos inesquecíveis de descontração. Obrigada a todos pelo enriquecimento acadêmico durante todo este período de bolsa. É uma honra trabalhar com pessoas tão dedicadas e amigas como vocês.

A todos da família LPRAD, pela contribuição em meu trabalho, em especial, a você Marcelino... meu muito obrigada.

Não poderia esquecer dos professores Eurípides e Walter Barra, que sempre acreditaram em meu potencial como aluna, desde os primeiros momentos de graduação. Meu eterno agradecimento a vocês, que para mim e minha querida BP11, são muito mais que professores, são verdadeiros educadores.

Por fim, deixo meus agradecimentos a todos que contribuíram, direta ou indiretamente, para a realização deste trabalho.

RESUMO

Este trabalho apresenta uma abordagem baseada em soluções de *middleware*, para prover interoperabilidade em sistemas heterogêneos. É proposta uma metodologia para a implementação e avaliação do desempenho de sistemas, utilizando técnica de aferição de dados, a partir da inserção de soluções de *middleware* em um cenário real. A aquisição dessas informações ou medidas de desempenho serão obtidas por meio de coleta de dados reais extraídos do sistema em estudo, possibilitando assim, realizar um diagnóstico mais preciso sobre o comportamento deste, e ainda, observar a viabilidade de se implementar soluções que promovam tal interoperabilidade, levando em consideração um dos fatores mais importantes quando se trata da programação de *middleware* : eficiência x transparência. O estudo de caso foi realizado no Laboratório de Eletromagnetismo Aplicado (LEA), da Universidade Federal do Pará (UFPA), onde foi aplicada a interoperabilidade em um ambiente cliente/servidor. Neste cenário, foram analisadas as duas soluções de *middleware* mais referenciadas na literatura especializada atualmente, RMI (*Remote Method Invocation*) e CORBA (*Common Object Request Broker Architecture*). Suas arquiteturas e suas principais características são descritas. Em seguida são apresentadas comparações detalhadas do desempenho dessas soluções de *middleware* em um sistema heterogêneo real.

Palavras-chaves: Interoperabilidade, *Object Request Broker* , Avaliação de Desempenho em Sistemas Computacionais.

ABSTRACT

This work presents an approach based on solutions of middleware to provide interoperability in real heterogeneous systems. It is proposed a methodology for the implementation and evaluation of the performance of computer systems, using the technique of gauging of data, by the insertion of solutions of middleware in a real scene. The information and performance data will be acquired by means of collection of real data extracted from the system under study, thus, it is possible to do a more complete diagnosis of the system behavior. Besides, it is observed the viability of implementing solutions that promote such interoperability taking into account the factor most important to the middleware programming: efficiency x transparency. The study case was done at the Laboratory of Applied Electromagnetism (LEA) which belongs to the Federal University of Pará (UFPA), where the interoperability was applied in an server/client environment. In this scenario, it was analyzed the two most important middleware solutions in the current literature, RMI (Remote Method Invocation) and CORBA (Common Object Request Broker Architecture). Its architectures and its main characteristics are described. After that, detailed comparisons of the middleware solutions performance in real a heterogeneous system are presented.

Keywords: Interoperability, Object Request Broken, Performance Evaluation in Computer Systems .

*“O Futuro independe do Passado dado o
Presente”.*

Markov

Sumário

Lista de Tabelas	8
Lista de Figuras	9
1 INTRODUÇÃO	11
1.1 JUSTIFICATIVA	11
1.2 MOTIVAÇÃO	11
1.3 RELEVÂNCIA DO ESTUDO	12
1.4 ESTRUTURA DO TRABALHO	12
2 MIDDLEWARE	14
2.1 CONSIDERAÇÕES INICIAIS	14
2.2 PADRONIZAÇÃO	15
2.2.1 Object Request Broker	15
2.3 INTEROPERABILIDADE POR MEIO DE WEB SERVICES	28
2.3.1 XML	29
2.3.2 SOAP	30
2.4 CONSIDERAÇÕES FINAIS	31
3 TRABALHOS CORRELATOS	32
3.1 CONSIDERAÇÕES INICIAIS	32
3.2 TRABALHOS	32
3.3 CONSIDERAÇÕES FINAIS	35
4 ESTUDO DE CASO	37

4.1	CONSIDERAÇÕES INICIAIS	37
4.2	DESCRIÇÃO DO EXPERIMENTO	37
4.3	OBJETIVOS DO ESTUDO	38
4.4	AMBIENTE DE TESTES	38
4.4.1	Cenário	38
4.4.2	Equipamentos Utilizados	40
4.5	AVALIAÇÃO DE DESEMPENHO A PARTIR DE TÉCNICA DE AFERIÇÃO DE DADOS	41
4.6	TESTES EFETUADOS	42
4.6.1	Interfaces dos Servidores Remotos	42
4.6.2	Implementação das Aplicações Cliente e Servidor	43
4.7	ANALISE DOS RESULTADOS OBTIDOS	44
4.7.1	Análise do Desempenho da Rede	44
4.7.2	Análise dos Recursos de Hardware	58
4.8	CONSIDERAÇÕES FINAIS	61
5	CONCLUSÕES	63
5.1	CONTRIBUIÇÕES DO TRABALHO	63
5.2	DIFICULDADES ENCONTRADAS	64
5.3	TRABALHOS FUTUROS	65
	Referências Bibliográficas	66
	Apêndices	70
A	Códigos Fontes	70
B	Diagramas de Classe	104

Lista de Tabelas

4.1	Descrição dos Elementos do Cenário	39
4.2	Configurações das Máquinas Utilizadas	40

Lista de Figuras

2.1	Comunicação entre duas aplicações por meio do <i>middleware</i>	14
2.2	<i>Middleware Object Request Broker</i> [6].	16
2.3	Arquitetura RMI	19
2.4	Arquitetura OMA	23
2.5	Arquitetura CORBA	24
4.1	Cenário onde foi realizado o estudo de caso	38
4.2	Gráfico comparativo da vazão da rede sem/durante a inserção das soluções <i>middleware</i>	45
4.3	Gráfico comparativo do consumo da banda sem/durante a inserção das soluções <i>middleware middleware</i>	47
4.4	Gráfico comparativo do jitter sem/durante a inserção das soluções <i>middleware</i>	48
4.5	Gráfico comparativo do jitter sem/durante a inserção das soluções <i>middleware</i>	49
4.6	Gráfico comparativo do jitter sem/durante a inserção das soluções <i>middleware</i>	50
4.7	Gráfico comparativo do jitter sem/durante a inserção das soluções <i>middleware</i>	51
4.8	Gráfico comparativo do jitter sem/durante a inserção das soluções <i>middleware</i>	52
4.9	Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções <i>middleware</i>	53
4.10	Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções <i>middleware</i>	54

4.11	Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções <i>middleware</i>	55
4.12	Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções <i>middleware</i>	56
4.13	Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções <i>middleware</i>	57
4.14	Exemplo de monitoração dos recursos de hardware	58
4.15	Consumo dos recursos do <i>hardware</i> pelo <i>middleware</i> RMI	60
4.16	Consumo dos recursos do <i>hardware</i> pelo <i>middleware</i> CORBA	61
B.1	Diagrama de Classes - <i>Middleware</i> RMI - Servidor e Cliente	105
B.2	Diagrama de Classes - <i>Middleware</i> CORBA - Servidor	106
B.3	Diagrama de Classes - <i>Middleware</i> CORBA - Cliente	107

1 INTRODUÇÃO

1.1 JUSTIFICATIVA

A utilização das redes de computadores e a popularização dos microcomputadores possibilitaram a construção de sistemas compostos por um grande número de processadores, interligados por redes de alta velocidade, em contraste com os sistemas centralizados, compostos por um único processador, memória e periféricos.

Atualmente, motivação como economia, distribuição geográfica de empresas, tolerância a falhas, crescimento incremental, apontam para a descentralização das aplicações. Por outro lado, a descentralização poderá ocasionar algumas “desvantagens”. A principal delas é o software: existe a necessidade de um novo software, mais complexo e sobre o qual não existe um consenso referente aos padrões tecnológicos a serem utilizados para desenvolvê-lo. Outra preocupação é a heterogeneidade, que pode se apresentar de diferentes formas: rede, sistema operacional, arquitetura de hardware, linguagem de programação.

Para se criar uma integração entre tecnologias heterogêneas há a necessidade de utilizarsoluções de interoperabilidade. Essas soluções incluem estratégias de implementação de acordo com cada problema estudado ou identificado. Comumente, essas soluções são denominadas de *middleware*, que consiste em uma camada de comunicação entre os componentes que desejam se comunicar e compartilhar recursos.

1.2 MOTIVAÇÃO

A heterogeneidade dos sistemas distribuídos, baseados em objetos, impõe a necessidade de especificações abertas, com interfaces padronizadas, levando ao desenvolvimento de *middleware*. As tecnologias de *middleware* emergiram como uma parte crítica da infraestrutura da tecnologia de informação [1]. Os sistemas de comunicações atuais, tais como computadores e redes de telecomunicações, são compostos por uma coleção de dispositivos heterogêneos cujas aplicações necessitam interagir. Os sistemas de *middleware* são usados mascarar a heterogeneidade de aplicações compostas por diversos componentes

distribuídos que funcionam em sistemas de redes e telecomunicações distintos.

1.3 RELEVÂNCIA DO ESTUDO

As abordagens baseadas em implementações de *middleware* provêm a interoperabilidade em sistemas heterogêneos. Entretanto, via de regra, essa integração não é antecipada por um planejamento, o que pode vir a comprometer a eficiência da solução [2]. Para isso, se faz necessário um estudo prévio sobre as principais características e funcionalidades das plataformas de *middleware*s no cenário atual, levando em consideração fatores determinantes como: portabilidade e escalabilidade.

Este trabalho aborda os dois sistemas de objetos distribuídos atualmente mais referenciados nas literaturas especializadas e largamente utilizados, são eles:

- *Java remote Method Invocation*(RMI);
- *Common Object Request Broker Architecture*(CORBA).

Cada solução de *middleware* possui suas próprias características. Tais características foram investigadas de forma comparativa a fim de se obter o melhor desempenho em um sistema heterogêneo real. Neste contexto, foram abordadas medidas importantes na avaliação de desempenho de sistemas, como: vazão, consumo da banda, jitter, perda de pacotes e consumo dos recursos de hardware.

Em linhas gerais, o objetivo do trabalho consistiu em adicionar à bibliografia especializada um estudo comparativo, no âmbito da avaliação de desempenho de redes e sistemas distribuídos, entre estas duas importantes plataformas de *middleware*.

1.4 ESTRUTURA DO TRABALHO

O trabalho encontra-se dividido conforme as subseções abaixo:

- No capítulo 2, é realizada uma abordagem teórica referente à tecnologia de *middleware*, capaz de prover uma forma de compartilhamento de recursos e troca de informações entre sistemas heterogêneos. Em especial, neste capítulo, é feita uma

abordagem voltada para as tecnologias de sistemas distribuídos baseadas em objetos, considerando vantagens e desvantagens das mesmas.

- No capítulo 3, é destinado aos trabalhos correlatos, apresentando trabalhos em que sejam aplicadas as soluções de interoperabilidade; apontando também trabalhos que abordem as soluções para os problemas do processo de integração das soluções de interoperabilidade nos sistemas distribuídos.
- No capítulo 4, como estudo de caso, será apresentada a aplicação da metodologia baseada em um cenário real. As medidas de desempenho obtidas a partir dos testes comparativos entre as duas plataformas de *middleware*, são interpretadas e analisadas.
- No capítulo 5 são apresentadas as contribuições e dificuldades na realização desta pesquisa, assim como sugestões para futuros trabalhos na área.

2 MIDDLEWARE

2.1 CONSIDERAÇÕES INICIAIS

Quando se deseja realizar a integração entre sistemas heterogêneos, é necessário buscar estratégias de implementação de soluções factíveis para o problema em questão. Essas estratégias, via de regra, são implementadas por meio da utilização de uma tecnologia denominada *middleware*. *Middleware* consiste em uma camada de software que provê uma abstração de programação, assim como o mascaramento da heterogeneidade de redes de computadores, hardware, sistemas operacionais e linguagens de programação [3]. Desta forma, *middleware* é uma camada de software responsável por mascarar a complexidade da integração em sistemas heterogêneos [2]. A figura 2.1 ilustra estes conceitos.

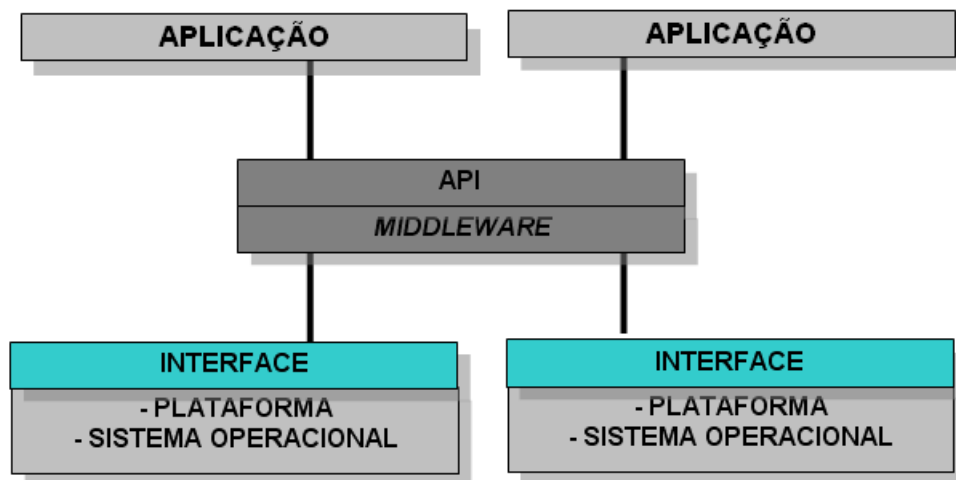


Figura 2.1: Comunicação entre duas aplicações por meio do *middleware*

Middleware se refere à integração de sistemas de qualquer porte. Podendo existir de várias formas e em vários níveis de ambientes. Toda forma de aplicação, linguagem de programação, sistema operacional e hardware tem sido alvo para uma integração utilizando soluções e alternativas de middleware [2].

Há inúmeros exemplos de soluções envolvendo *middleware* bem sucedidos. Porém é importante notar que enquanto um *middleware* facilita a diversidade e a interoperabilidade entre heterogeneidade, eles não resolvem o problema por completo. De

uma maneira geral, a maior parte das formas de *middleware* tende a reduzir a complexidade pela introdução de uma homogeneidade artificial nos sistemas. No entanto, essa “homogeneidade forçada”, via de regra, insere certo atraso (*delay*) da comunicação, devido às colisões entre os sistemas heterogêneos [4].

Idealmente todos os componentes de um sistema interoperável deveriam ser igualmente rápidos, ricos em funcionalidades e expressivos em seu modelo de dados [5]. Na prática, todos esses fatores não são usualmente possíveis devido a uma série de escolhas sobre a alternativa a ser implementada, dependendo do grau de homogeneidade do sistema.

Um ponto importante na utilização e funcionamento dos *middlewares* consiste em sua padronização, estratégia importante para auxiliar os usuários a selecionarem *middleware* baseados em qualidade.

2.2 PADRONIZAÇÃO

A padronização consiste em uma estratégia necessária para que o mercado de componentes de software se desenvolva. Em linhas gerais, ela se constitui na definição de interfaces para permitir a composição entre componentes e na definição de modelos e arquiteturas para permitir a interoperabilidade entre sistemas.

Existem vários tipos de *middlewares*, contudo a classificação relevante para este trabalho consiste em *middlewares* orientados a objetos. Baseados fortemente em tecnologia e protocolos de objetos, esses padrões objetivam promover um ambiente transacional distribuído e heterogêneo, onde códigos transformados em objetos são usados para compor transações. Além da transparência de localização e de chamada de código heterogêneo e distribuído, este tipo de *middleware* também oferece uma série de serviços (na forma de objetos abstratos ou interfaces) que enriquecem o ambiente e facilitam sua utilização. Suas principais características são detalhadas na seção abaixo.

2.2.1 Object Request Broker

Object Request Broker, ou simplesmente ORB, é uma tecnologia que gerencia a comunicação e a troca de dados entre objetos. Em outras palavras, o ORB provê interoperabilidade em sistemas de objetos distribuídos. Ele permite a construção de sistemas pelo

agrupamento de objetos que se comunicam entre si através dele. Os detalhes da implementação do ORB geralmente não são importantes para os desenvolvedores de sistemas distribuídos. Os desenvolvedores devem se preocupar apenas com os detalhes da interface do objeto.

A tecnologia ORB permite a comunicação de objetos entre diferentes máquinas, diferentes softwares e diferentes fornecedores. Um ORB provê um diretório de serviços e auxilia a estabelecer conexões entre clientes e estes serviços. Esta definição é ilustrada na Figura 2.2.

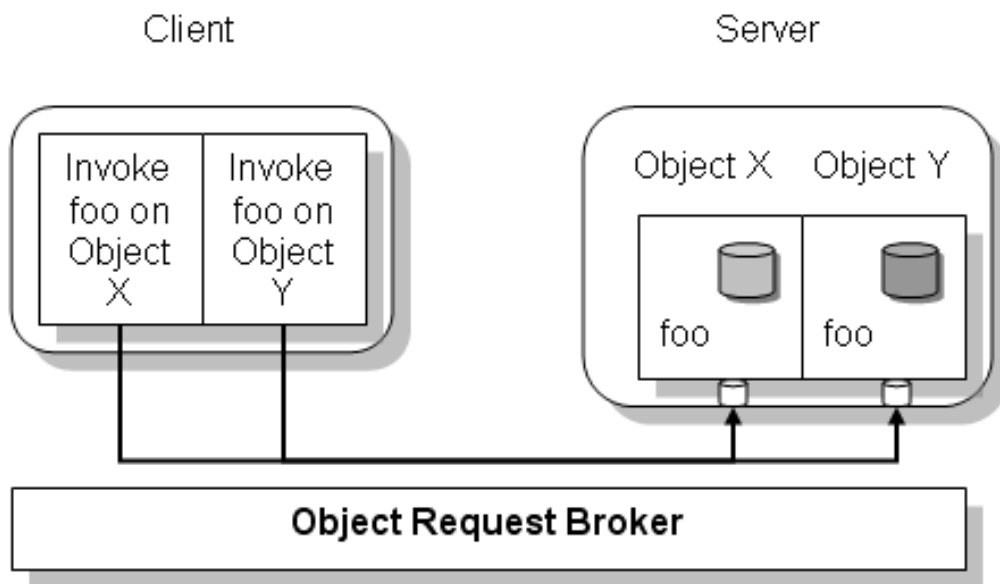


Figura 2.2: *Middleware Object Request Broker* [6].

É importante ressaltar que um ORB deve suportar muitas funções com o objetivo de operar de forma eficiente, e, além disso, muitas dessas funções são transparentes ao usuário do ORB. Ainda, é de responsabilidade do ORB fornecer a transparência de localidade, ou seja, fazer com que o objeto requisitado pareça local para o cliente, enquanto que na realidade ele se localiza em um processo ou máquina em qualquer ponto da rede, por exemplo. Portanto, um ORB fornece um barramento para a comunicação entre objetos em diferentes sistemas. Este é o primeiro passo para alcançar a interoperabilidade em sistemas de objetos distribuídos.

É importante ressaltar ainda, que o ORB permite aos objetos esconder seus detalhes de implementação dos clientes. Isto inclui linguagens de programação, sistemas operacionais, hardware, e localização de objetos. Cada um desses itens deve ser pensado

como uma transparência, e diferentes tecnologias de ORBs podem suportar diferentes transparências, fazendo com que os benefícios da orientação a objetos entre plataformas e canais de comunicação sejam estendidos [6].

Além disso, um ORB age como um intermediário para as requisições que os clientes enviam para os servidores. É responsável por todos os mecanismos requeridos para encontrar a implementação de um objeto, preparar a implementação para receber a requisição, e comunicar os dados na requisição. Ele também utiliza as informações da requisição para determinar a melhor implementação que satisfaça o pedido. Estas informações podem incluir: a operação que o cliente está requisitando, o tipo de objeto que será executado, e qualquer informação adicional armazenada no contexto do objeto que está sendo requisitado.

Os principais padrões para computação distribuída que hoje utilizam a tecnologia ORB são:

- RMI - *Remote Method Invocation*
- CORBA - *Common Object Request Broker Architecture*
- DCOM - *Distributed Component Object Model*

O Java *Remote Method Invocation* (RMI) e a *Common Object Request Broker Architecture* (CORBA) consistem nos dois sistemas de objetos distribuídos mais importantes e largamente utilizados atualmente. O padrão DCOM, desenvolvido pela Microsoft®, apesar de ser uma solução aberta, opera somente na plataforma Windows®. É uma tecnologia excelente para interação de sistemas sobre esta plataforma, sem interação com outros sistemas operacionais. Para que clientes UNIX façam conexões a servidores Microsoft®, por exemplo, são necessários softwares adicionais, complexos de serem gerenciados e o inverso não é permitido. Isso caracteriza uma grande desvantagem deste padrão em relação a outras soluções existentes [6].

Cada solução possui suas próprias características e estão sendo utilizadas na indústria para várias aplicações que vão desde a área comercial até os cuidados com a saúde. Neste trabalho serão focadas as tecnologias RMI e CORBA, que consistem nos padrões de maior utilização como soluções de interoperabilidade.

2.2.1.1 RMI

RMI - *Remote Method Invocation* - é uma tecnologia de programação de objetos distribuídos que é parte integrante da plataforma Java® e está disponível desde a versão 1.1 do Java® SDK (*Sun Development Kit*). RMI® permite que métodos de objetos remotos sejam chamados da mesma forma que métodos de objetos locais, estendendo o conceito de programação distribuída para a linguagem Java. RMI® possibilita a comunicação entre objetos rodando em máquinas virtuais diferentes, independentes dessas estarem na mesma máquina física ou não [7].

A arquitetura de RMI® é baseada na definição de interface do serviço remoto. Um objeto define o serviço que ele provê através de uma interface, que é usada pelo cliente para fazer uma chamada ao serviço.

Baseando-se na definição de uma interface implementa-se a classe que representa o objeto servidor. A partir da implementação do objeto servidor geram-se as classes do stub do cliente e do skeleton do servidor, utilizando-se a ferramenta *rmic* (RMI *Compiler*) - compilador de interfaces de Java RMI® [8].

Tanto a implementação do objeto servidor quanto o stub do cliente implementam a interface do serviço remoto. Quando o cliente executa uma chamada a um método da interface, o stub do cliente é invocado. Em seguida o stub do cliente se comunica com o skeleton do servidor para que a chamada da implementação real do método seja invocada [8].

A partir da versão 1.2 do Java SDK, *skeletons* não são mais necessários. Um protocolo adicional de *stubs* foi introduzido do lado servidor, responsável pela funcionalidade realizada anteriormente pelos *skeletons*. Entretanto, *skeletons* ainda podem ser gerados se for necessária a compatibilidade com versões anteriores de SDK [8].

2.2.1.1.1 Arquitetura

A arquitetura do sistema RMI é dividida em camadas, de forma a facilitar sua expansão e inclusão de novas funcionalidades. A figura 2.3 ilustra a arquitetura RMI® [7].

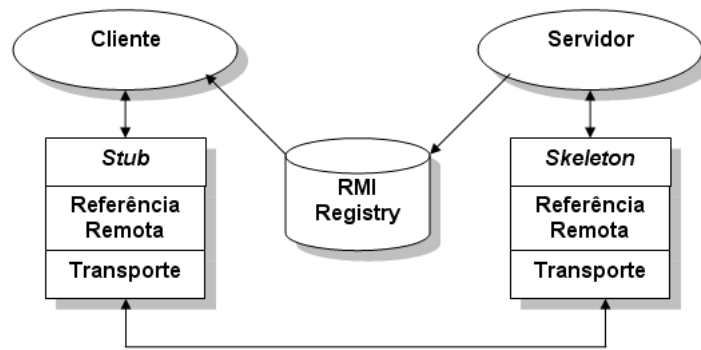


Figura 2.3: Arquitetura RMI

- Camada de *Stubs & Skeletons* - essa camada reside logo abaixo da aplicação. Como seu próprio nome indica, ela inclui o *stub* do cliente e o *skeleton* do servidor. O *stub* do cliente é responsável por empacotar (*marshal*) o nome do método remoto e os argumentos da chamada e repassar a chamada ao objeto servidor, funcionando como um *proxy* do objeto remoto. Os serviços da camada inferior (Referência Remota) são utilizados para realizar essa função. O *skeleton* do servidor é responsável por receber a chamada vinda do cliente, desempacotá-la (*unmarshalling*) e chamar o método remoto propriamente dito. Após a execução do método remoto o *skeleton* empacota os dados a serem devolvidos (argumento de saída, código de retorno do método ou exceção levantada durante a execução) e os envia para o cliente.
- Camada de Referência Remota - do lado do cliente, essa camada é responsável pela criação de uma referência para o objeto da máquina remota (estabelecendo uma conexão com o objeto) e por disponibilizar os mecanismos para a invocação de métodos remotos - utilizando os serviços da camada inferior (Transporte). Do lado do servidor, essa camada recebe os dados da chamada e os repassa a chamada para o *skeleton* do servidor. A camada de referência remota é responsável por todas as semânticas envolvidas no estabelecimento de uma conexão, tais como ativação de objetos remotos dormentes, gerenciamento de tentativas de conexão (*connection retry*), etc.
- Camada de Transporte - responsável pela conexão entre as máquinas virtuais do cliente e do servidor e pelo transporte das mensagens entre as mesmas. Essa camada utiliza o protocolo JRMP (*Java Remote Method Protocol*) sobre TCP/IP para a comunicação entre as máquinas virtuais. JRMI é um protocolo proprietário da Sun

baseado em fluxos de *bytes* (*streams*). O protocolo HTTP também pode ser utilizado para encapsular as mensagens do protocolo JRMP quando existe alguma máquina *firewall* entre o cliente e o servidor.

2.2.1.1.2 Serviço de Localização de Servidor (Serviço de Nomes ou Diretório)

Para invocar um método remoto, é necessária uma referência para um objeto remoto. Esta referência pode ser obtida como código de retorno de uma chamada a outro método remoto ou através do serviço de localização de servidor (serviço de nomes ou diretórios). O servidor deve publicar o nome do serviço após a criação do objeto que o implementa [7].

RMI® possui um serviço de localização de servidores específico chamado RMI® *Registry*, mas outros serviços podem ser usados, tais como JNDI - *Java Naming and Directory Interface* [8].

2.2.1.1.3 Características

A seguir são listadas algumas das principais características de Java RMI® [7].

- Serialização de Objetos (*Object Serialization*) - RMI® usa serialização de objetos para transformar tipos primitivos e objetos em um formato apropriado para transmissão de uma máquina virtual para outra (empacotamento dos dados ou *marshalling*) [7].
- Passagem de Parâmetros - em Java® parâmetros são sempre passados por valor se as chamadas de métodos ocorrem dentro da mesma máquina virtual, tanto para tipos primitivos quanto para referências a objetos. O mesmo é válido para códigos de retorno de métodos. Em RMI®, em uma chamada a um método remoto (ou seja, um método de um objeto de uma outra máquina virtual), os seguintes tipos de passagens de parâmetros (e códigos de retorno de métodos) podem ocorrer:
 - Tipos primitivos são passados por valor;
 - Objetos locais são passados por valor, assim como todos os objetos referenciados pelo mesmo - ou seja, todo o grafo de dependência a partir do objeto é seriado, formando uma estrutura de dados auto-contida (sem referências a endereços de memória locais), para ser transmitido de uma máquina virtual para outra;

- Objetos remotos são passados por referência - entretanto, uma referência a um objeto remoto é convertida para uma referência para o respectivo *stub*.
- Polimorfismo Distribuído - RMI® não “trunca” nenhum tipo de dados passado de uma máquina virtual para outra, ou seja, RMI® provê polimorfismo distribuído. Por exemplo, se um objeto de uma classe A é esperado como parâmetro de um método e ao invés disso é passado um objeto de uma classe B, subclasse de A, a definição completa do objeto é enviada e não apenas a parte que corresponde à classe A.
- Carga Dinâmica de Classes/*Stubs* (*Dynamic Class/Stub Download*) - se uma classe de um objeto passado como parâmetro ou devolvido por um método não estiver carregada na máquina virtual, RMI® tenta carregar a implementação da classe localmente. Caso a implementação não esteja disponível na máquina local, RMI® tenta obtê-la a partir da localização específica pela propriedade *codebase* (*java.rmi.server.codebase* - lista de URLs), passando juntamente como objeto na chamada ou retorno do método. O mesmo procedimento é válido quando um cliente obtém uma referência para um objeto remoto e o respectivo *stub* não está disponível na máquina local [9].
- Coleta de Lixo Distribuída - RMI® estende o conceito de coleta de lixo para programação distribuída. Entretanto, não é possível precisar se um objeto remoto não tem mais nenhuma referência ou se as conexões com os clientes que o referenciam forem interrompidas. Sendo assim, RMI® considera que um objeto remoto é candidato para a coleta de lixo distribuída quando ele não é acessado após um determinado período de tempo.
- Segurança - RMI® utiliza o mesmo mecanismo usado em Java para a segurança, baseado na utilização de objetos gerenciadores de segurança (classe *SecurityManager*). Um objeto *SecurityManager* é necessário em qualquer máquina virtual que precise obter a implementação de uma classe remotamente. O *SecurityManager* define quais operações podem ser executadas por objetos de classes obtidas remotamente.
- Ativação de Objetos Remotos - RMI® provê mecanismo para que objetos remotos sejam criados sob demanda, ou seja, apenas quando um cliente tenta acessar um

de seus métodos. Alguns tutoriais nas páginas de RMI® contêm maiores detalhes sobre as formas de ativação de objetos remotos [10].

2.2.1.2 CORBA

CORBA - *Common Object Request Broker Architecture* - é uma arquitetura para a programação de objetos distribuídos que especifica o componente ORB da OMA - Object Management Architecture (*Object Request Broker*) [11], [12], [13], [14]. A organização responsável pela criação, definição e evolução da OMA é a OMG - Object Management Group, um consórcio formado por aproximadamente 800 membros, dentre eles empresas de pequeno e grande porte, universidades e institutos de pesquisa [15]. É importante destacar que a OMG é responsável pela especificação da OMA e de seus componentes, não pela sua implementação.

Existem diversas implementações de CORBA disponíveis no mercado: produtos comerciais - tais como *Orbix* da IONA [16] e *Visibroker* da Borland [17] - e implementações abertas - por exemplo, Java *IDL* [18] e *JacORB* [19].

Assim como RMI®, o modelo adotado por CORBA é baseado no modelo de objetos, onde os serviços disponibilizados por objetos servidores, são acessados através de uma interface bem definida pelos objetos clientes. Para a definição de interfaces, CORBA especifica uma linguagem denominada OMG IDL (*Interface definition language*) e mapeamentos dessa linguagem para determinadas linguagens de programação, tais como C, C++, Java, Smalltalk e COBOL. A partir da definição da interface de um objeto em IDL, um compilador IDL é utilizado para geração do *stub* do cliente e do *skeleton* do servidor.

As aplicações cliente e servidor podem ser desenvolvidas utilizando esses stubs e skeletons. Entretanto, CORBA provê mecanismos para chamadas remotas de procedimento nos quais cliente e servidor não necessitam ter conhecimento da definição de interfaces em tempo de compilação [7].

Implementações da arquitetura CORBA como as mencionadas acima normalmente incluem implementações do elemento Serviços de Objetos (*Object Services*) da arquitetura OMA tais como Serviço de Nomes (*Name Service*), Serviço de Eventos (*Event Service*), etc.

2.2.1.2.1 Arquitetura OMA

A arquitetura OMA é baseada no Modelo de Objetos (*Object Model*) e no Modelo de Referências (*Reference Model*). O Modelo de Objetos é um modelo cliente/servidor, no qual o objeto servidor provê serviços aos objetos clientes.

Esses serviços são chamados através de operações definidas na interface do objeto servidor [7].

O Modelo de Referência define os seguintes componentes, assim como suas interfaces e as interações entre os mesmos. A figura 2.4 ilustra a arquitetura OMA[7].

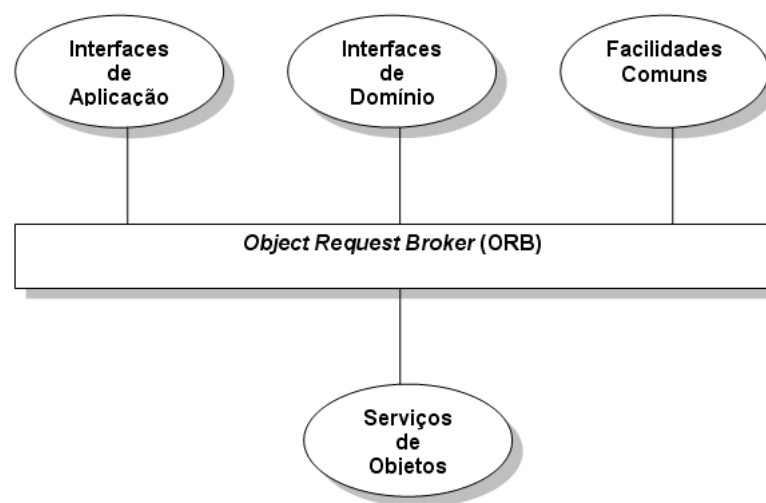


Figura 2.4: Arquitetura OMA

- ORB (*Object Request Broker*) - consiste no componente central da arquitetura OMA, responsável pela comunicação entre objetos distribuídos. A ORB provê transparência de localização, ativação e independência de linguagem de programação entre cliente e servidor.
- Serviços de Objetos (*Object Services*) - esse componente define os serviços básicos utilizados por quaisquer aplicações: nomes, eventos, ciclo de vida, persistência, transações, controle de concorrência, relacionamento, externalização, busca, propriedades, seguranças, entre outros.
- Facilidades Comuns (*Commons Facilities*) - são interfaces orientadas para aplicações de usuário, que podem ser utilizadas por aplicações de domínios distintos: interface com o usuário, gerenciamento de informação, gerenciamento de sistemas e gerenciamento de tarefas.

- Interfaces de Domínio (*Domain Interfaces*) - são interfaces orientadas para domínios específicos, tais como: finanças, medicina, manufatura e telecomunicações.
- Interfaces de Aplicação (*Application Interfaces*) - são interfaces não padronizadas desenvolvidas para as aplicações específicas. Essas interfaces utilizam os serviços dos demais componentes.

2.2.1.2.2 Arquitetura CORBA

A arquitetura CORBA especifica as interfaces e os serviços que o componente ORB da arquitetura OMA deve fornecer. CORBA define os seguintes componentes [7],

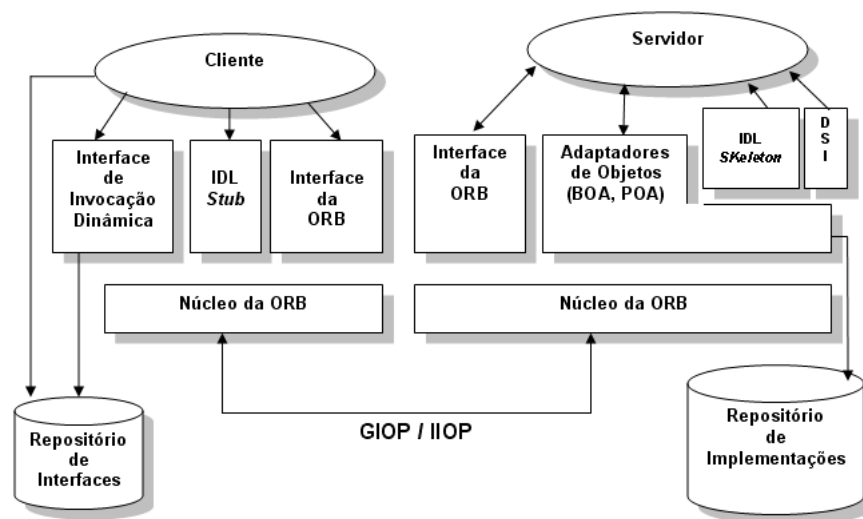


Figura 2.5: Arquitetura CORBA

- Núcleo da ORB (*ORB core*) - responsável pela comunicação entre clientes e servidores, provê mecanismo para a chamada a método de objetos remotos, implementando transparência de localização e de ativação de servidores, assim, como independência de linguagem de programação e de plataforma de hardware.
- OMG IDL (*Interface Definition Language*) - é uma linguagem declarativa utilizada para a especificação das operações e atributos de objetos CORBA. IDL permite também a definição de módulos, contendo definições de diversos objetos. A OMG especifica mapeamento de IDL para algumas linguagens de programação, implementados por compiladores IDL.
- IDL *Stubs* - gerados por compiladores IDL, stubs são utilizados por aplicações cliente para chamadas a métodos de servidores remotos. Os *stubs* são responsáveis pela

criação de chamadas e pelo envio das mesmas ao servidor, através da ORB. No retorno das chamadas os stubs repassam os argumentos de saída e códigos de retorno recebidos para a aplicação cliente.

- IDL *Skeleton* - gerados por compiladores IDL, skeletons são utilizados por aplicações servidores no tratamento de chamadas de métodos remotos. Os skeletons recebem as chamadas de métodos através da ORB e as repassam para o objeto destino. *Skeletons* também são responsáveis por enviar os argumentos de saída e códigos de retorno dos métodos para a aplicação cliente.
- Adaptadores de Objetos (*Object Adapters*) - responsáveis por prover os mecanismos e interfaces necessárias para que implementações de objetos CORBA utilizem os serviços da ORB. Sua funcionalidade inclui os seguintes serviços: registro de objetos, geração de referência a objetos, ativação e desativação de objetos e servidores, e registro de implementações. A arquitetura CORBA atualmente especifica dois tipos de adaptadores.
 - BOA (*Basic Object Adapter*) - primeiro adaptador especificado pela OMG, de definição muito genérica em algumas áreas, foi implementado de formas incompatíveis por diferentes produtos, dificultando a portabilidade das implementações de objetos CORBA;
 - POA (*Portable Object Adapter*) - criado para resolver problemas de portabilidade entre implementações de adaptadores de diferentes produtos. Inclui também algumas funcionalidades adicionais: suporte a objetos transientes e persistentes, ativação de objetos explícita e sob demanda, especificação de políticas de multithreading, segurança e gerenciamento de objetos, entre outras.
- Interface para Invocação Estática (SII - *Static Invocation Interface*) - é a invocação de métodos remotos através da utilização de stubs de clientes e *skeletons* de servidores, gerados pelo compilador IDL, ou seja, toda a informação necessária para a chamada é conhecida em tempo de compilação. É importante notar que a utilização de stubs pelas aplicações cliente não implica na utilização de *skeletons* nos servidores e vice-versa, ou seja, interfaces dinâmicas (descritas a seguir) podem ser utilizadas independentemente no cliente e no servidor.

- Interface para Invocação Dinâmica (DII - *Dynamic Invocation Interface*) - é a invocação de métodos remotos pela qual informações a respeito dos mesmos (nome, parâmetros e respectivos tipos) é obtida em tempo de execução por meio da utilização do Repositório de Interfaces. Dessa forma uma aplicação cliente pode invocar os métodos de quaisquer objetos CORBA, se a necessidade de conhecer suas interfaces em tempo de compilação.
- Interface de Esqueleto Dinâmica (DSI - *Dynamic Skeleton Interface*) - é o equivalente a DII, mas do lado do servidor. Esse mecanismo permite a implementação de servidores capazes de tratar chamadas de métodos sem a necessidade de se utilizar *skeletons* gerados por compiladores IDL.
- Repositório de Interfaces (*Interface Repository*) - é um repositório que provê informações sobre interfaces de objetos remotamente registrados. Permite que aplicações cliente e servidor obtenham definições de interfaces de objetos CORBA em tempo de execução.
- Repositório de Implementações (*Implementation Repository*) - é um repositório que contém informações utilizadas pela ORB para localizar e ativar objetos servidores.
- Protocolo GIOP (*Generic Inter ORB Protocol*) - é um protocolo de comunicação que garante a interoperabilidade entre implementações de ORBs distintas. Esse protocolo define as mensagens e o formato de representação dos dados para a comunicação entre ORBs. A implementação de GIOP sobre TCP/IP é chamada de IIOP (*Inter-ORB Protocol*).

2.2.1.2.3 Características

CORBA é uma tecnologia que se desenvolveu ao longo de vários anos [20]. Entre as principais características de CORBA pode-se citar:

- Independência de Linguagem de Programação - a utilização de uma linguagem específica para a definição de interfaces (OMG IDL) e a existência de mapeamentos dessa linguagem para diferentes linguagens de programação possibilitam a comunicação entre aplicações cliente servidor escritas em linguagens distintas.
- Independência de Plataforma de *Hardware* - proporcionada pela disponibilidade de implementação de CORBA em plataformas distintas.

- Interoperabilidade entre Implementações Distintas - a utilização do protocolo GIO/IOP permite que aplicações cliente e servidor, desenvolvidas utilizando-se implementações distintas de CORBA, se comuniquem transparentemente.
- IOR (*Interoperable Object Reference*) - é um formato padrão para referências a objetos, necessário para a interoperabilidade entre diferentes implementações de ORBs. Contém informações que permitem que uma ORB localize e se comunique com objetos remotos.
- Invocação Dinâmica de Métodos - realizada através da utilização do repositório de interfaces e da interface para invocação dinâmica.
- Modos de Passagens de Parâmetros - OMG IDL permite que se especifiquem parâmetros de entrada (*in*), de saída (*out*) ou de entrada e saída (*inout*).
- Objetos Passados por Valor (*Objects by Value*) - possibilidade de passar e devolver objetos por valor em uma chamada remota de procedimento, através da utilização do elemento *valuetype* de OMG IDL.
- Modos de Invocação de Métodos Remotos:
 - Síncrono - a aplicação cliente fica bloqueada esperando a resposta da invocação do método remoto;
 - Síncrono postergado (*deferred synchronous*) - a aplicação cliente faz a chamada e não é bloqueada. Para obter a resposta a aplicação pode fazer uma chamada específica para verificar se a resposta já está disponível (e então recebê-la) ou bloquear a sua execução enquanto a resposta não é recebida. Esse modo só pode ser utilizado com invocação dinâmica, pois a especificação para geração de interfaces de invocação estática através de *stubs* não contempla esse modo de invocação;
 - Sem resposta (*one-way*) - nesse modo não existe resposta. A aplicação cliente invoca o método remoto e a ORB faz o possível para que a chamada seja executada, mas não há garantia (*best effort*);
 - Assíncrona com *callback* - a aplicação cliente não é bloqueada ao fazer a chamada ao método remoto. Nesse modo uma referência a um objeto é passada como parâmetro adicional, utilizada pela ORB para a entrega da resposta;

- Assíncrono com *polling* - este modo é similar ao modo síncrono postergado. A aplicação cliente faz a chamada ao método remoto e recebe imediatamente no retorno da chamada um objeto desenvolvido por valor (*valuetype*). Esse objeto pode ser utilizado posteriormente para verificar se a resposta já está disponível ou bloquear a execução da aplicação enquanto a resposta não é recebida. Os modos assíncronos de invocação foram especificados posteriormente [20], juntamente com alterações na especificação da geração de *stubs*, podendo ser utilizados com invocação estática de métodos remotos.
- Persistência - CORBA permite que a criação de objetos transientes, cujo tempo de vida é associado ao tempo de vida da aplicação que os hospedam, e os objetos persistentes, que não possuem essa limitação. Maiores detalhes sobre o uso de servidores transientes e persistentes em Java IDL podem ser encontrados em [20].
- Tipo Genérico de *any* - é um tipo abstrato de dados de OMG IDL que pode conter valores de qualquer outro tipo OMG IDL.
- Segurança - a OMG especifica diferentes mecanismos para a configuração e utilização de segurança em CORBA. Esses mecanismos provêm controle de acesso, autenticação, comunicação segura etc. maiores detalhes podem ser encontrados em [21].
- CORBA não possui nenhum mecanismo de coleta de lixo distribuída, basicamente porque nem todas as linguagens de programação que podem ser utilizadas possuem essa característica.

2.3 INTEROPERABILIDADE POR MEIO DE WEB SERVICES

A crescente popularização da internet e o crescimento do número de serviços e informações disponibilizadas por meio desta, nos leva a repensar a forma como nossos sistemas de informação são desenvolvidos. Diversos *middleware* de comunicação, como CORBA e RMI® , vêm se adaptando a este novo cenário. Outras tecnologias também estão surgindo como peças-chaves para prover a interoperabilidade no contexto atual, dentre elas, se

destaca a utilização de *web services* como soluções de *middleware* para integrar diversas aplicações.

Web services, no sentido geral do termo, significam serviços oferecidos pela *internet*. Trata-se de uma nova promessa de revolução/evolução das tecnologias de informação, com o objetivo de mudar a forma como os aplicativos compartilham o comportamento (lógica de negócio) e os dados entre si [22].

O padrão *web services* fornece uma maneira aberta para integrar aplicações, tanto dentro como fora das organizações. O uso de padrões abertos torna possível a integração de componentes e aplicações de forma independente da tecnologia utilizada. Sua principal vantagem em relação às demais tecnologias baseados em objetos, é a simplicidade e a utilização de padrões abertos, não proprietários [23].

Uma importante vantagem do *web services* é que eles intencionalmente acomodam diversidade e heterogeneidade, não somente em aplicações, sistemas operacionais e *hardware*, mas também na relação com outros sistemas de *middleware* [4]. Um caminho para se pensar sobre *web services* é que eles podem ser apontados como um “*middleware de middleware*”. Dado que sistemas mais estáveis têm estado fragilizados por suas incapacidades de incorporar novas funcionalidades e integração com outros sistemas.

Os *web services* representam a evolução de alguns padrões e protocolos de larga utilização e utilizados para criar a internet como a conhecemos hoje. Sua infraestrutura é construída sobre os padrões e tecnologias abertas, devidamente definidas e de ampla aceitação, como XML (*eXtensible Markup Language*), SOAP (*Simple Object Access Protocol*), WSDL (*Web Service Description Language*) e UDDI (*Universal Description, Discovery and Integration*). Sua implementação é possível e portátil sobre qualquer sistema operacional e linguagem de programação que suporte estes padrões. Os padrões XML e SOAP consistem nos principais utilizados atualmente e são melhores detalhados nas seguintes subseções.

2.3.1 XML

A linguagem de marcação extensível, é um subconjunto da SGML (*Standard Generalized Markup Language*) e é utilizada para resolver muitos problemas relacionados a padrões utilizados para transferência de dados entre empresas e representa a base da comunicação

entre os web services.

XML é um tipo da meta-linguagem, isto é, ela não é uma linguagem de programação e pode ser considerado como em um conjunto de regras que podem ser usadas para formatar textos a fim de estruturar e padronizar os dados. A vantagem principal de XML é que é extensível, isto é, cada desenvolvedor pode construir seu próprio padrão de documento, além de independente plataforma. XML é similar ao HTML no que tange ao uso de marcadores e atributos. No HTML, os marcadores e os atributos associados são usados para apresentar o texto em um determinado formato por um navegador. Visto que, no caso de XML, os marcadores usados para organizar um conjunto de dados e da interpretação destes dados são deixados inteiramente à aplicação que a lê, eventualmente, interpretando sua informação. Apenas como exemplo, quando `< p >` no HTML indicar um parágrafo, em XML pode informar o preço, o parâmetro, a pessoa, etc., dependendo de um determinado contexto [22].

Um documento XML pode ou não possuir um documento validador de tipos (*Data Types Definition* - DTD). O DTD descreve a estrutura de um conjunto de documentos similares e promove uma padronização baseados em suas regras de validação para o documento e os tipos de objetos. Porém, para validar dados mais complexos, refinando assim os tipos de dados, o *World Wide Consortium* (W3C) tem desenvolvido uma especificação XML Schema que provê mais funcionalidades do que as existentes no DTD original. Podendo suportar mais tipos de dados e modelagens de relacionamentos [24].

2.3.2 SOAP

SOAP ou *XML Protocol* é um protocolo simples, comercialmente independente que provê mecanismos leves para intercâmbio de informações estruturadas entre diferentes entidades, em um ambiente distribuído, usando XML. Define um formato padrão para codificação de dados e uma estrutura simples para expressar a semântica da aplicação por meio de um mecanismo modular de empacotamento de informações e codificação dos tipos de dados contidos nos pacotes. Permite a sua utilização em combinação com uma variedade de outros protocolos de comunicação, entretanto, na maioria das aplicações, normalmente ele é utilizado em combinação com o HTTP. Pelo fato de utilizar protocolos de comunicação extremamente conhecidos e difundidos, e por isso registrados nas estruturas de segurança disponíveis na internet (*firewalls*), as mensagens SOAP conseguem transpor

facilmente tais estruturas, sem que seja necessário qualquer esforço adicional por parte dos desenvolvedores e administradores de redes [24].

2.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou as alternativas de *middleware* mais referenciadas na literatura visando à integração de sistemas heterogêneos.

Foram apresentadas, principalmente, características gerais dessas tecnologias, além de algumas vantagens e dificuldades de cada uma.

Os objetivos principais de uma solução *middleware* são: a integração entre sistemas heterogêneos e a intermediação entre aplicações e o sistema operacional. Para que estes objetivos sejam alcançados, um *middleware* deve fornecer serviços que atendam ao domínio de aplicações para qual foi construído, sendo importante que este serviço tenha sua base em uma das API's ou protocolos padrões.

3 TRABALHOS CORRELATOS

3.1 CONSIDERAÇÕES INICIAIS

As soluções de *middleware* têm se tornando a mais atrativa forma de prover a interoperabilidade entre tecnologias heterogêneas. Neste capítulo serão apresentadas algumas dentre as inúmeras aplicações de soluções de *middleware*.

3.2 TRABALHOS

Dentre os trabalhos realizados na área, podem ser apontados como exemplos de aplicação:

Em [25] é desenvolvido um sistema de *middleware* para computação em grade oportunista, denominado InteGrade. Este, utiliza CORBA como sua infra-estrutura de objetos distribuídos, beneficiando-se de um substrato elegante e consolidado, o que se traduz na facilidade de implementação, uma vez que a comunicação entre os módulos do sistema é abstraída pelas chamadas de métodos remotas. O autor ressalta que CORBA também permite o desenvolvimento para ambientes heterogêneos, facilitando a integração de módulos escritos nas mais diferentes linguagens, executando sobre diversas plataformas de hardware e software. E ainda que CORBA fornece uma série de serviços úteis e consolidados [25], como os serviços de Transações, Persistência, Nomes e *Trading*, os quais podem ser utilizados pelo InteGrade, facilitando assim o seu desenvolvimento.

O trabalho desenvolvido em [26] é identificar e evidenciar alguns conceitos de interoperabilidade que poderiam emergir no desenvolvimento de aplicações CORBA distribuídas e heterogêneas para WWW, é proposto um *framework* para a evolução da interoperabilidade para integração de sistemas distribuídos desenvolvidos na arquitetura CORBA com a WWW (CORBA-WWW). O *framework* provê um conjunto de critérios para sistemas de interoperabilidade, podendo servir como template para fazer comparações de sistemas desenvolvidos em diversas soluções de *middleware*. Os critérios sugeridos pelo framework são: ininterrupto, segurança, confiabilidade, escalabilidade e desempenho.

Já em [27], tem como objetivo realizar a interoperabilidade entre os dispositivos móveis (por exemplo, celular, PDA) e dispositivos pessoais de localização (por exemplo, GPS) a nível de protocolos e aplicações, dispondo ainda de uma interface única para as aplicações de localização dos dispositivos, assim os dispositivos móveis e pessoais de localização seriam capazes de exercer a mesma funcionalidade de localização, tornando os custos diferenciados tanto para dispositivos quanto aplicações. Para efetivar essa integração é proposta uma arquitetura que utiliza um protocolo chamado de MLP (*Mobile Location Protocol*) baseado em XML, que permite a interoperabilidade de sistemas e de serviços de localização transportado por http ou https, desenvolvido pela LIF (*Location Interoperability Forum*). A arquitetura implementada é dividida em três blocos para efetivar sua adaptabilidade e flexibilidade: o ProxyMLP, os serviços de tradução e os blocos de técnicas de localização específicas. Cada um desses blocos possuem suas regras bem definidas, sendo o mais importante deles o ProxyMLP que por sua vez é o responsável pela conexão com um banco de dados de informações de localização e é o único que faz uma interface para aplicações externas, o serviço de tradução tem a funcionalidade de traduzir as mensagens transmitidas pelas técnicas de localização e enviá-las a uma interface interna do MLP (ProxyMLP). Algumas características e objetivos da arquitetura descritas no trabalho são: criação de uma arquitetura escalável e distribuída; criação de uma interface única para aplicações externas que integre todo suporte e sistemas de localização internos; criação de um ambiente que se adaptem com mínimo impacto as diferentes técnicas de localização; criação de rotinas de segurança que permita não somente a proteção da informação mas também o acesso de aplicações externas não autorizadas.

Em [28] é proposto um *middleware* orientado a serviços para redes de sensores sem fio baseado em XML e SOAP. Segundo [28], a escolha de XML foi motivada pelo seu alto poder de representatividade e sua capacidade de extensão. A linguagem oferece flexibilidade suficiente para representar consultas e tarefas de sensores. SOAP foi escolhido por ser um protocolo leve e versátil. Além disso, ambos são padrões de facto na *Web*. Os autores afirmam que a maior parte dos usuários de RSSFs (Redes de Sensores Sem Fio) não está interessada apenas em obter os dados extraídos da rede através de uma interface gráfica, mas utilizá-los para alimentar suas próprias aplicações. Esse tipo de interface aplicação-aplicação é altamente beneficiado pela abordagem de *middleware* baseado em XML e SOAP.

Segundo os autores, o principal benefício do *middleware* proposto está no fato

do sistema oferecer uma camada de interoperabilidade entre as diferentes aplicações e a rede de sensores. O *middleware* cria a abstração de uma RSSF genérica, que fornece serviços para as várias aplicações. Além disso, é ressaltado pelos autores que através da utilização da linguagem XML e do protocolo SOAP, o *middleware* proposto naturalmente possibilita a interoperabilidade da RSSF com a Internet. Desta forma, as funcionalidades da RSSF podem ser disponibilizadas como Serviços Web, que podem ser acessados por qualquer aplicação usuário com acesso a Internet, de modo independente de linguagem e plataforma.

Em [29] e [30] há dois trabalhos comparativos que analisam o desempenho das soluções de *middlewares* em sistemas heterogêneos reais.

No primeiro, os autores concentram-se na comparação de desempenho entre RMI(Java SDK 1.14) e CORBA (Visigenic Visibroker for Java 3.0). O objetivo foi medir o custo adicional dessas tecnologias nas chamadas de métodos remotos.

Três cenários foram utilizados: cliente e servidor na mesma máquina, cliente e servidor em computadores separados e servidor rodando em um computador com clientes sendo executados simultaneamente em 2, 3, 4, 5, 6, 7 e 8 computadores. Em todos os cenários foi utilizada uma rede com pouca interferência de outros tipos de tráfego. A comparação entre o primeiro cenário e o segundo cenário teve o intuito de determinar o custo adicional da comunicação em rede, enquanto que a comparação entre o segundo e o terceiro cenários teve como objetivo determinar a degradação de desempenho em um ambiente com vários clientes simultâneos.

Foram utilizados métodos com códigos de retorno simples (todos os tipos primitivos de Java®) e métodos desenvolvendo cadeias de caracteres. Uma ferramenta de análise de código foi utilizada com o objetivo de identificar os métodos que consumiram o maior tempo de execução.

Os resultados obtidos determinaram que nenhuma das duas tecnologias é consideravelmente mais rápida ou mais lenta do que a outra. Em cenários simples, como métodos devolvendo tipos primitivos ou cadeias de caracteres pequenas com poucos clientes simultâneos, RMI® obteve um desempenho melhor. Nos casos onde vários clientes foram utilizados ao mesmo tempo CORBA demonstrou ser uma tecnologia mais robusta. Para cadeias grandes, RMI® obteve também um melhor desempenho.

Já no segundo trabalho, além de uma comparação de desempenho, foi efe-

tuada uma comparação de usabilidade e das características de diversas tecnologias de programação distribuída em Java: Java RMI (1.4.2_01), CORBA (JavaIDL 1.4.2_01 e JacORB 1.4.1), Java/ICE (1.1.1) e Java/SOAP (Apache Axis 1.1).

Os métodos testados foram divididos em dois grupos: métodos com códigos de retorno e parâmetros simples (tipos primitivos de Java) e métodos com parâmetros e códigos de retorno do tipo de *bytes* de diversos tamanhos.

Embora três cenários tenham sido utilizados (cliente e servidor na mesma máquina, cliente e servidor em máquinas diferentes - rede local e Internet), os resultados de apenas um deles foram apresentados, pois o autor concluiu que as razões entre as diferenças de tempo foram mantidas em todos os cenários.

A análise de usabilidade efetuada indicou que a tecnologia mais simples de ser utilizada é Java RMI. CORBA e ICE empataram em segundo lugar, com SOAP sendo considerada a tecnologia de maior dificuldade de utilização.

Em termos de desempate, para métodos simples CORBA (JacORB) foi pouco melhor do que RMI. ICE ficou em terceiro lugar e SOAP ocupou a última posição, com um desempenho muito inferior às demais tecnologias. Para métodos desenvolvendo vetores de *bytes* de diversos tamanhos, ICE obteve o melhor desempenho, seguido de perto por Java RMI e CORBA. SOAP novamente teve o pior desempenho entre as tecnologias analisadas.

3.3 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados alguns trabalhos e linhas de pesquisa em diferentes domínios de aplicações de soluções de interoperabilidade; demonstrando algumas possíveis utilizações das soluções de *middleware* e o melhoramento das integrações das soluções no processo de adaptação dos sistemas distribuídos, além de sua aplicabilidade e eficiência nas mesmas.

Foram vistos trabalhos utilizando várias soluções e melhoramento da interoperabilidade que vão desde a implementação de *middleware* para a computação em grade, utilizando CORBA para o desenvolvimento do sistema [25]; implementação de solução de CORBA para Internet [26]; interoperabilidade em dispositivos móveis [27] até soluções que propuseram um *middleware* orientado a serviços para redes de sensores sem fio baseado em XML e SOAP [28].

Também foram vistos dois trabalhos comparativos, [29] e [30], que investigam o desempenho entre diversas soluções de *middleware*.

4 ESTUDO DE CASO

4.1 CONSIDERAÇÕES INICIAIS

Chamadas de procedimentos remotos são mais complexas do que chamadas de procedimentos locais, dado que as primeiras envolvem comunicação entre processos distintos (tipicamente rodando em máquinas diferentes) e o empacotamento (*marshalling*) /desempacotamento das chamadas e dos valores desenvolvidos pelas mesmas em um formato independente de plataforma e adequado para essa comunicação [7].

Neste trabalho é apresentada uma comparação de desempenho das duas principais tecnologias de programação distribuída apresentadas no capítulo 2. A metodologia proposta para a avaliação do sistema consiste na coleta de dados, importante técnica de aferição para avaliação do desempenho de sistemas.

Este capítulo descreve a metodologia adotada nas comparações e contém detalhes sobre os programas de teste desenvolvidos. Em seguida são descritos os resultados obtidos nos testes comparativos.

4.2 DESCRIÇÃO DO EXPERIMENTO

O experimento realizado consiste em uma aplicação cliente/ servidor. O servidor oferece serviço de banco de dados, o qual é acessado remotamente pelos clientes da rede local. As informações extraídas do banco de dados são convertidas para o formato de documento xml e posteriormente serializadas, para serem enviadas ao cliente para posterior análise. A conversão em formato de documento permite que os dados extraídos da base de dados sejam visualizados de forma independente da aplicação que os produziu, evitando assim, problemas comuns como dependência de uma plataforma específica, não-extensibilidade e dificuldades na adequação a diferentes tipos de usuários [31]. Todo o processo de extração e transferência de arquivos XML é realizado por meio de *middleware* a fim de melhorar a conectividade entre a aplicação cliente/servidor. E ainda, permitir que os clientes do sistema acessem o serviço no servidor sem a preocupação sobre diferenças entre usuários.

O estudo de caso foi realizado no Laboratório de Eletromagnetismo Aplicado (LEA), na Universidade Federal do Pará (UFPA). Para isso, se utilizou um sistema real onde foram feitas medidas de desempenho a partir da coleta de dados.

4.3 OBJETIVOS DO ESTUDO

Realizar um estudo comparativo sobre a inserção das duas principais tecnologias de middleware largamente utilizadas em sistemas distribuídos - RMI e CORBA - em um ambiente cliente/servidor, utilizando-se de coleta de dados para extrair as medidas necessárias à avaliação de desempenho.

4.4 AMBIENTE DE TESTES

4.4.1 Cenário

Para a comparação do desempenho entre as duas soluções de *middleware*, os programas testes desenvolvidos foram executados no seguinte cenário.

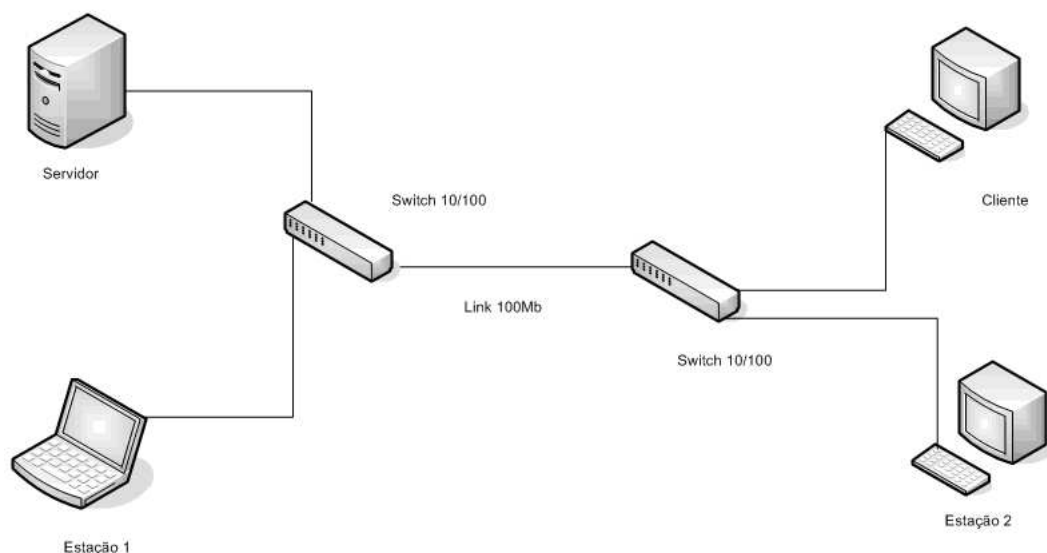


Figura 4.1: Cenário onde foi realizado o estudo de caso

A tabela 4.1 mostra a descrição dos principais elementos do cenário.

Tabela 4.1: Descrição dos Elementos do Cenário

Cenário	Descrição
Servidor	O servidor contém o banco de dados do fabricante MySQL® e executa as solicitações de serviços do cliente.
Cliente	Consiste no solicitante do serviço.
Estação 1 e 2	Estações de trabalho para caracterizar uma rede não dedicada em pleno funcionamento.

O cenário em estudo consiste em uma rede *Ethernet*, constituída por um servidor, um cliente e duas estações de trabalho. Todos os computadores da rede possuem sistemas operacionais independentes e suportam a mesma arquitetura de comunicação. Sendo assim, foi possível implementar um ambiente com diferentes arquiteturas de computadores e sistemas operacionais.

A interoperabilidade do sistema é garantida pela utilização da tecnologia de *middleware*. Assim, foram verificados os impactos da inserção dos respectivos padrões, RMI e CORBA, nos demais processos da rede. E ainda, a viabilidade da implementação desta tecnologia em uma ambiente cliente/servidor.

O cenário mostrado na figura 4.1 ilustra duas estações de trabalho. Essas estações são usadas para verificar o desempenho do sistema antes, durante e após a inserção do *middleware* na rede.

De acordo com o cenário proposto, as tarefas das soluções de *middleware*, em mais alto nível de abstração, consistem em:

- Conectar-se com a base de dados e gerar o documento XML;
- Transferir os documentos XML para o cliente solicitante.

A partir destes serviços realizados, observa-se então, a interferência no tráfego da rede e os recursos de hardware consumidos. Além disso, a comparação entre as tecnologias de *middlewares* neste cenário possibilita medir o custo adicional (*overhead*) da rede no desempenho da comunicação entre cliente e servidor.

4.4.2 Equipamentos Utilizados

As máquinas utilizadas para os testes são computadores da arquitetura x86 e suas configurações são descritas na tabela 4.2. Os concentradores da rede *Ethernet* utilizados são *switch* 10/100.

Tabela 4.2: Configurações das Máquinas Utilizadas

Máquinação	Processador	Memória	Placa de Rede	Sistema Operacional
Servidor	AMD Athlon XP 1.53 GHz	512 MB	VIA Compatable Fast Ethernet Adapter	Linux Ubuntu Kernel 2.6.17-10-386
Cliente	AMD Sem- prom 1.60 GHz	1 GB	Realtek RTL8169/8110 Gigabit Ethernet NIC	Linux Ubuntu Kernel 2.6.15-27-386
Estação 1	AMD Sem- prom 1.58 GHz	512 MB	SIS 900 Based PCI Fast Ethernet Adapter	Windows XP Profes- sional
Estação 2	Intel Pentium 4 1.60 GHz	1 GB	SIS 900 Based PCI Fast Ethernet Adapter	Linux Ubuntu Kernel 2.6.17-10-386

A utilização do sistema operacional Linux na máquina servidor possibilita a monitoração dos recursos de hardware a partir da requisição do cliente. A máquina cliente também utiliza a plataforma Linux para garantir uma maior conectividade com a máquina servidor. Todos os sistemas operacionais Linux possuem compilações de *Kernel* e periféricos diferentes, conforme visto na tabela 4.2. As estações de trabalho possuem sistemas operacionais diferenciados a fim de demonstrar uma rede heterogênea real. Elas caracterizam um ambiente não dedicado, isto é, uma rede local onde operam diversos serviços de rede, independentes da aplicação cliente/servidor.

4.5 AVALIAÇÃO DE DESEMPENHO A PARTIR DE TÉCNICA DE AFERIÇÃO DE DADOS

A avaliação de desempenho de sistemas computacionais consiste em um conjunto de técnicas e metodologias que permitem responder à questão de como se obter o melhor desempenho de um sistema computacional a um dado custo. O desempenho é um critério chave no projeto, na obtenção, e no uso dos sistemas computadorizados [32].

Para que seja efetuado um estudo de avaliação sobre um dado sistema, passos importantes devem ser considerados [32]. São eles:

- Definir os objetivos e limites do sistema;
- Selecionar as métricas;
- Selecionar a técnica de avaliação;
- Analisar e interpretar os dados;
- Refazer o estudo, se necessário;
- Apresentar os resultados.

Os objetivos do sistema consistem em analisar as condições da rede em si a partir da utilização de soluções de *middleware* para prover a interoperabilidade no ambiente heterogêneo. Os limites são observados a partir da investigação do ponto crítico da rede, isto é, a máxima interferência suportada pelo ambiente durante a inserção do *middleware* na rede. A aferição de dados foi selecionada como a técnica utilizada para avaliar o desempenho do sistema descrito na seção 4.4. Esta técnica trabalha com dados extraídos de sistemas reais, sendo assim, ela apresenta um alto grau de confiabilidade sobre o comportamento do sistema em estudo [32].

Para a realização da aferição sobre o sistema em questão, foi feito uma coleta de dados a partir das condições reais do sistema. Esta técnica de aferição é utilizada para avaliar sistemas já existente, para implementar um projeto de um novo sistema e ainda, para validar um determinado modelo. Ela provê resultados mais precisos, haja vista, que todas as análises são feitas a partir de dados coletados em um sistema real. Os dados extraídos do sistema foram armazenados em diversas amostras. Sobre estas

amostras foram aplicadas técnicas de avaliação de desempenho a fim de diagnosticar o comportamento do sistema a partir da inserção das soluções de *middleware*.

4.6 TESTES EFETUADOS

Como descrito anteriormente, os testes foram realizados no Laboratório de Eletromagnetismo Aplicado (LEA), os quais consistem em comparações entre chamadas de procedimentos remotos utilizando as duas tecnologias estudadas: RMI® e CORBA.

A linguagem de programação utilizada para a construção dos programas referentes à aplicação cliente/servidor foi a linguagem Java®, linguagem de construção da tecnologia RMI e da distribuição CORBA utilizada - JavaIDL.

A justificativa para a utilização da linguagem Java® está no fato de que ela consiste em uma linguagem multiplataforma, de fácil acesso e aprendizado, pois possui um reduzido número de construções. A diminuição das construções mais suscetíveis a erros de programação, tais como ponteiros e gerenciamento de memória via código de programação também faz com que a programação em Java® seja mais eficiente. Java® é uma linguagem computacional completa, adequada para o desenvolvimento de aplicações baseadas na rede Internet, redes fechadas ou ainda programas *stand-alone*. Ela contém um conjunto de bibliotecas que fornecem grande parte da funcionalidade básica da linguagem, incluindo rotinas de acesso à rede e criação de interface gráfica [33].

Durante a execução dos testes foi observado a utilização de CPU (Unidade Central de Processamento) e memória por meio do utilitário *top* do linux.

4.6.1 Interfaces dos Servidores Remotos

Os programas de testes desenvolvidos nas duas soluções de *middleware* implementam interfaces de serviços remotos equivalentes.

Em RMI a especificação de uma interface remota é equivalente à definição de qualquer interface em Java, a não ser pelos seguintes detalhes [34]:

- A interface deverá, direta ou indiretamente, estender a interface *Remote*;
- Todo método da interface deverá declarar que a exceção *RemoteException* (ou uma

de suas superclasses) pode ser gerada na execução do método.

Em CORBA, uma interface para um serviço é definida usando as construções da linguagem IDL (*Interface Definition Language*) por meio do aplicativo *idlj* que gera a Interface Java correspondente, com a tradução das construções IDL para as primitivas Java® segundo o padrão estabelecido em CORBA, e outros arquivos auxiliares [34].

4.6.2 Implementação das Aplicações Cliente e Servidor

As aplicações servidores, tanto em RMI ou CORBA, criam basicamente uma instância do objeto remoto e a registram utilizando um serviço de nomes apropriado.

As aplicações cliente desenvolvidas também são muito similares. A única diferença está na obtenção de uma referencia para o objeto remoto.

Em RMI, uma vez definida a interface remota e a classe que implementa o serviço remoto o próximo passo consiste em desenvolver o servidor RMI. Para isso, instancia-se um objeto que implementa os serviços oferecidos e os cadastra na plataforma de objetos distribuídos. O desenvolvimento do cliente RMI requer essencialmente a obtenção de uma referência remota para o objeto que implementa os serviços, isto deve-se a publicação previa destes pelo servidor. Uma vez obtida essa referência, a operação com o objeto remoto é indistinguível da operação com um objeto local [34].

Para que um serviço oferecido por um objeto possa ser acessado remotamente através de RMI, é preciso também que as classes auxiliares internas de *stubs* e *skeletons*, sejam criadas. Elas são responsáveis pela comunicação entre o objeto cliente e o objeto que implementa o serviço, sendo geradas durante a compilação da interface do servidor remoto.

Em CORBA, para a aplicação servidor, são criadas duas classes. A classe “Server” consiste no servidor que ativa o ORB (ver capítulo 2), cria o objeto que implementa o serviço, obtém uma referencia para o serviço de nomes e registra o objeto no diretório associado ao nome da classe. A classe “Servant” consiste na implementação do serviço especificado, ela é uma extensão do *skeleton* definido pelo aplicativo *idltojava*. A classe cliente por sua vez, ativa o ORB, obtém uma referencia para o serviço de nomes e, a partir deste serviço, obtém uma referência remota para o objeto com o serviço oferecido pelo servidor. Obtida esta referência, o método é invocado normalmente [34].

4.7 ANÁLISE DOS RESULTADOS OBTIDOS

4.7.1 Análise do Desempenho da Rede

Para cada tecnologia, os métodos definidos na interface do serviço remoto foram executados diversas vezes no cenário descrito na seção 4.4 a fim de se obter os valores médios entre as amostras. Para a coleta dos dados necessários para efetuar as medidas de desempenho foi utilizado o *software* de monitoração do desempenho de redes *Iperf.1.7.0*. O *software* estabelece a comunicação entre as duas estações de trabalho. A estação 1 transmite pacotes para a estação 2, esta por sua vez, armazena os dados referentes à transmissão em um arquivo de trace para posterior análise. A coleta de dados se concentra nas seguintes métricas:

- Vazão da Rede;
- Consumo da Banda;
- Jitter;
- Perda de Pacotes.

Para avaliar as métricas vazão e consumo da banda se utilizou a transmissão TCP entre as duas estações de trabalho. Já para as métricas jitter e perda de pacotes, foi utilizada a transmissão UDP. Nos testes UDP a taxa de transmissão foi variada de 10Mb a 50Mb a fim de obter uma comparação mais precisa sobre as métricas investigadas. É sabido que a rede opera até a faixa de 100Mb, porém a taxa de 50Mb foi a maior taxa de transmissão suportada pela rede local, devido as limitações do *software* de aferição.

Sobre as amostras se estabeleceu uma análise das condições da rede sem a carga gerada pelo *middleware* e a partir da carga gerada por este. As medições foram realizadas e comparadas entre as duas tecnologias estudadas a fim de estabelecer um diagnóstico sobre os impactos da utilização destas em um sistema semelhante ao descrito no cenário apresentado na seção 4.4.

Sobre os dados obtidos foram calculadas as médias para as amostras da transmissão TCP e UDP, respectivamente, bem como os respectivos desvios padrão. As medias foram utilizadas nas comparações de desempenho de cada tecnologia enquanto que os desvios padrão serviram de referência para a análise dos resultados obtidos. Sobre estes

valores foi calculado o intervalo de confiança a fim de estabelecer o nível de confiança de cada resultado.

Abaixo seguem os resultados comparativos para cada métrica citada anteriormente. Eles visam apresentar um panorama de como o sistema se comporta em função da inserção do *middleware* na rede não dedicada.

4.7.1.1 Vazão

A vazão corresponde a taxa na qual os pedidos são atendidos (servidos) pelo sistema.

É apresentado o gráfico da vazão dos pacotes na figura 4.2. Nesta, observa-se o comportamento da rede sem a carga gerada pelo *middleware* e a partir da inserção deste na rede.

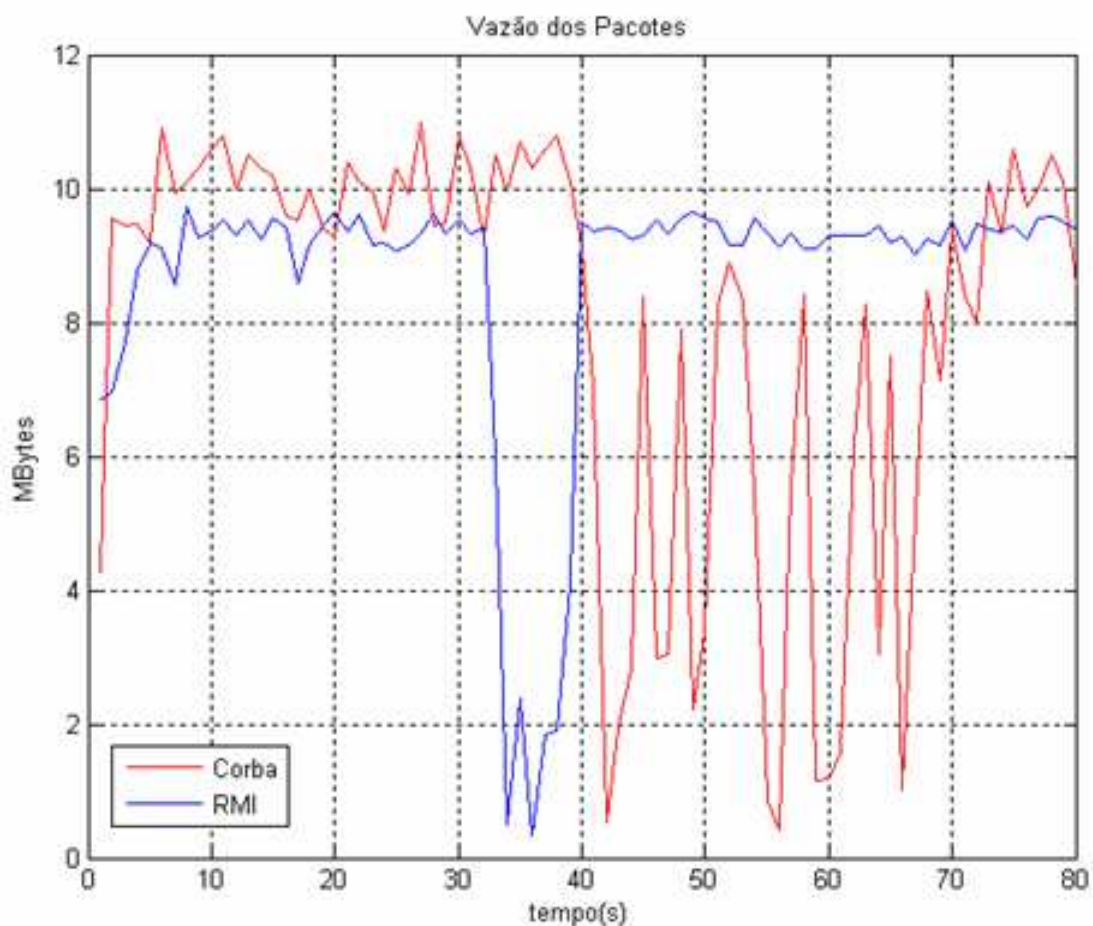


Figura 4.2: Gráfico comparativo da vazão da rede sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;

- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundos.

De acordo com a figura 4.2 observa-se que a rede possui uma vazão média entre 9MB a 11MB. Com a inserção do *middleware* RMI, a vazão da rede que possuía um valor em torno de 9MB sem a carga gerada pelo *middleware*, assume valores contínuos entre 0.4MB a 2MB durante a operação do *middleware* na rede. Já com *middleware* CORBA, a vazão da rede que antes assumia valores em torno de 10MB, apresenta valores que vão de 0.5MB a 8.5MB durante a transmissão. Assim, temos que durante toda a operação de RMI a vazão da rede assume baixos valores, enquanto que durante a operação de CORBA a vazão da rede só assume baixos valores durante a comunicação entre cliente e servidor propriamente dita, isto é, somente quando o servidor responde as requisições do cliente. Durante as demais operações a rede apresenta uma vazão consideravelmente boa.

Este fato se dá porque, neste caso, RMI não possui uma “gerência de rede” como o *middleware* CORBA, isto é, a partir do momento em que o cliente requisita o servidor, RMI utiliza o máximo possível da rede para manter a comunicação entre ambos, mesmo que não haja nenhuma troca de informações entre eles, como por exemplo, durante o processo de serialização das informações da Base de Dados, prejudicando assim, os demais processos de rede.

Assim, observamos que a vazão da rede assume um percentual de queda máximo de -86% de acordo com RMI, e percentual de queda máximo de -95% de acordo com CORBA, respectivamente.

4.7.1.2 Consumo da Banda

Com base nesta métrica é possível observar o percentual de consumo da banda da rede com a inserção das soluções de *middleware*. O gráfico comparativo é mostrado na figura 4.3.

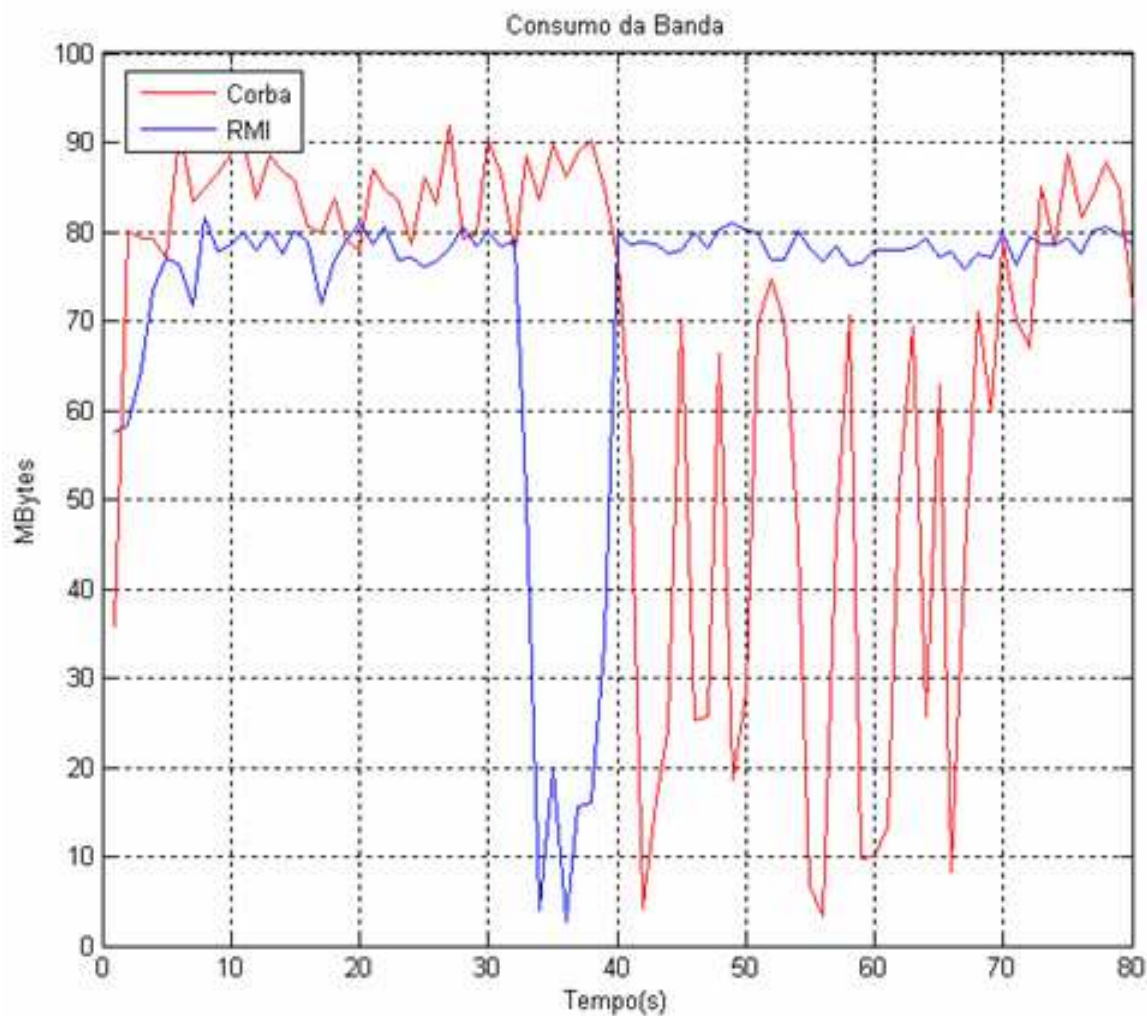


Figura 4.3: Gráfico comparativo do consumo da banda sem/durante a inserção das soluções *middleware middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundos.

O gráfico da figura 4.3 segue a mesma análise de comportamento do gráfico 4.2. Ambos ressaltam o não “gerenciamento da rede” por parte de RMI. A rede apresenta uma largura de banda média entre 78MB a 90MB. O percentual máximo do consumo de banda com a inserção de RMI é de 77%. Já com o *middleware* CORBA o consumo da banda máximo ficou em torno de 80%.

4.7.1.3 Jitter

O jitter mostra a variação estatística do atraso na rede. Como citado anteriormente, esta métrica foi avaliada com cinco taxas de transmissão diferentes: 10, 20, 30, 40 e 50Mb. A seguir são mostrados os gráficos respectivos a cada taxa de amostragem diferente.

4.7.1.3.1 Taxa - 10 Mb

Na figura 4.4 temos o gráfico comparativo do jitter para a taxa de transmissão 10Mb.

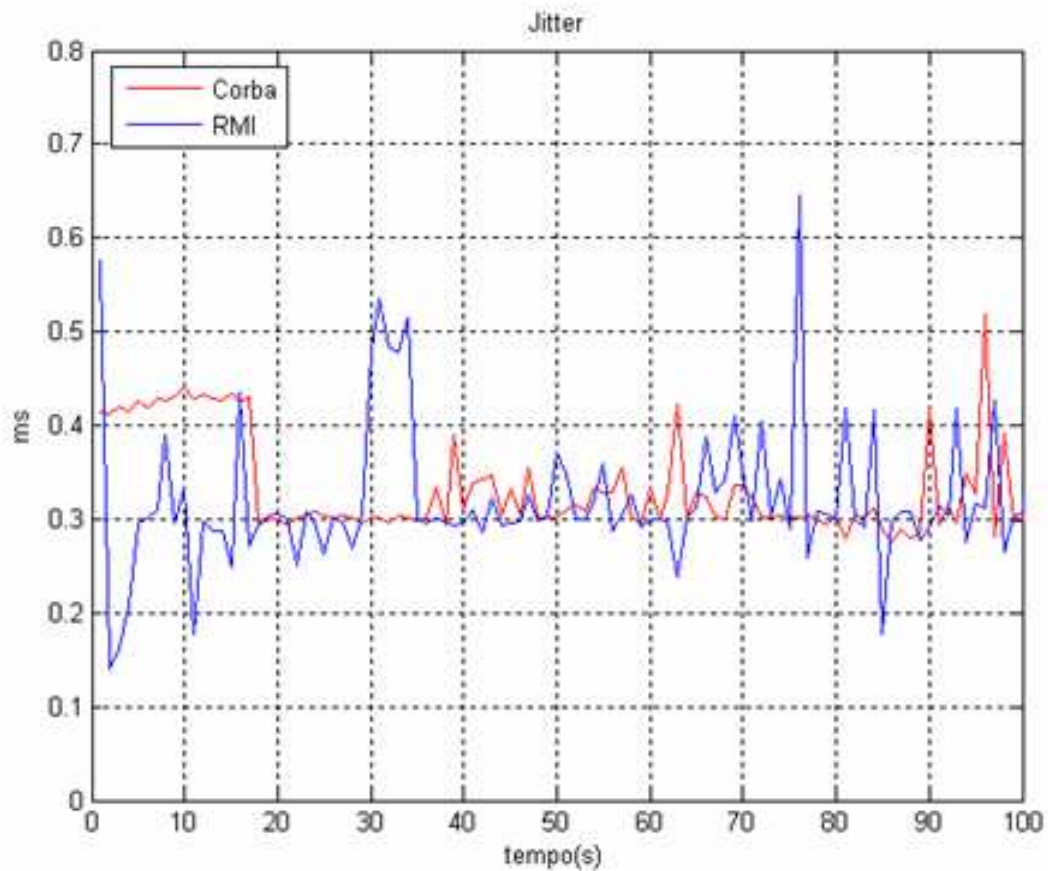


Figura 4.4: Gráfico comparativo do jitter sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

De acordo com o gráfico observamos que as duas tecnologias apresentam uma significativa variação de atraso, porém o *middleware* RMI causou uma variação de atraso mais elevada

durante sua transmissão em relação ao CORBA, caracterizando um pior desempenho nesta taxa de transmissão.

4.7.1.3.2 Taxa - 20 Mb

Na figura 4.5 temos o gráfico comparativo do jitter para a taxa de transmissão 20Mb.

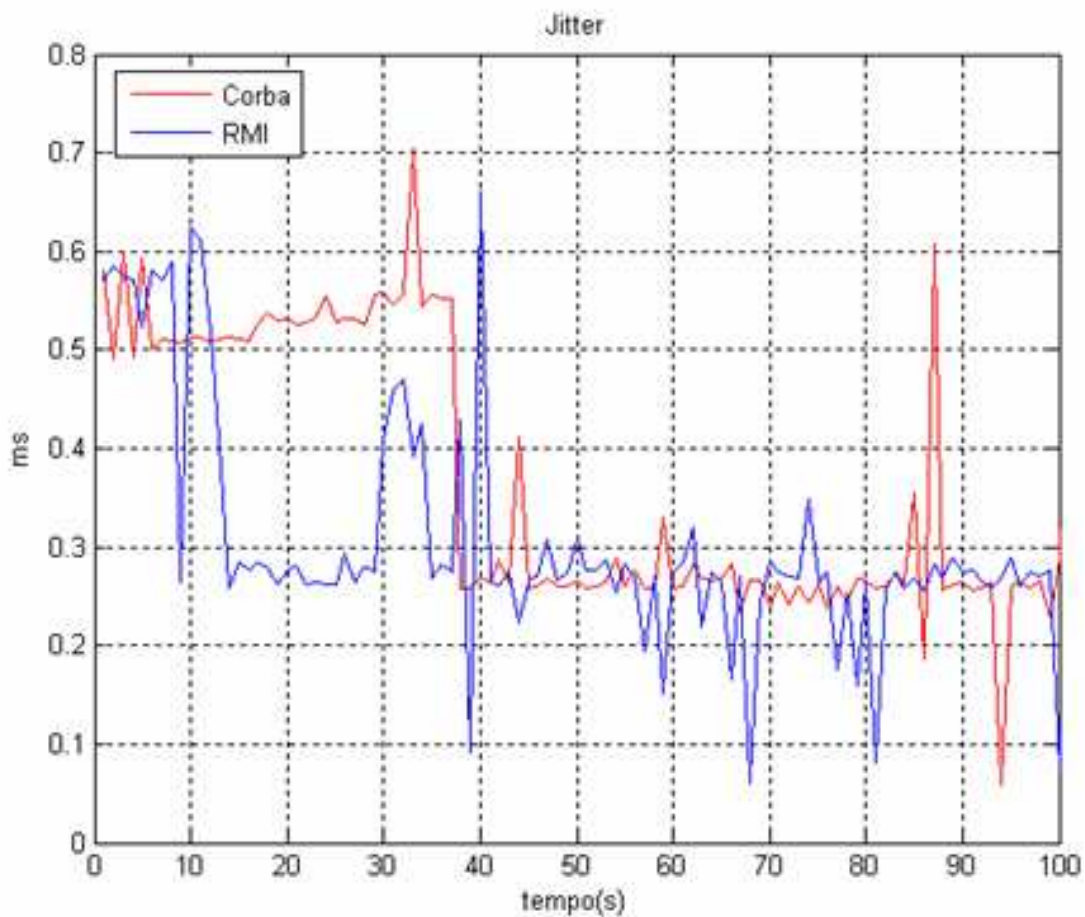


Figura 4.5: Gráfico comparativo do jitter sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

Nesta taxa de transmissão observamos também uma variação de atraso significativa para as duas tecnologias. As condições iniciais de transmissão entre as duas estações de trabalho mostram alguns processos de rede interferindo nesta transmissão. Durante a inserção das soluções, RMI apresentou novamente um maior jitter na transmissão, mas observamos

alguns picos elevados durante a transmissão de CORBA. Em média, RMI obteve um jitter menos elevado durante esta taxa de transmissão.

4.7.1.3.3 Taxa - 30 Mb

Na figura 4.6 temos o gráfico comparativo do jitter para a taxa de transmissão 30Mb.

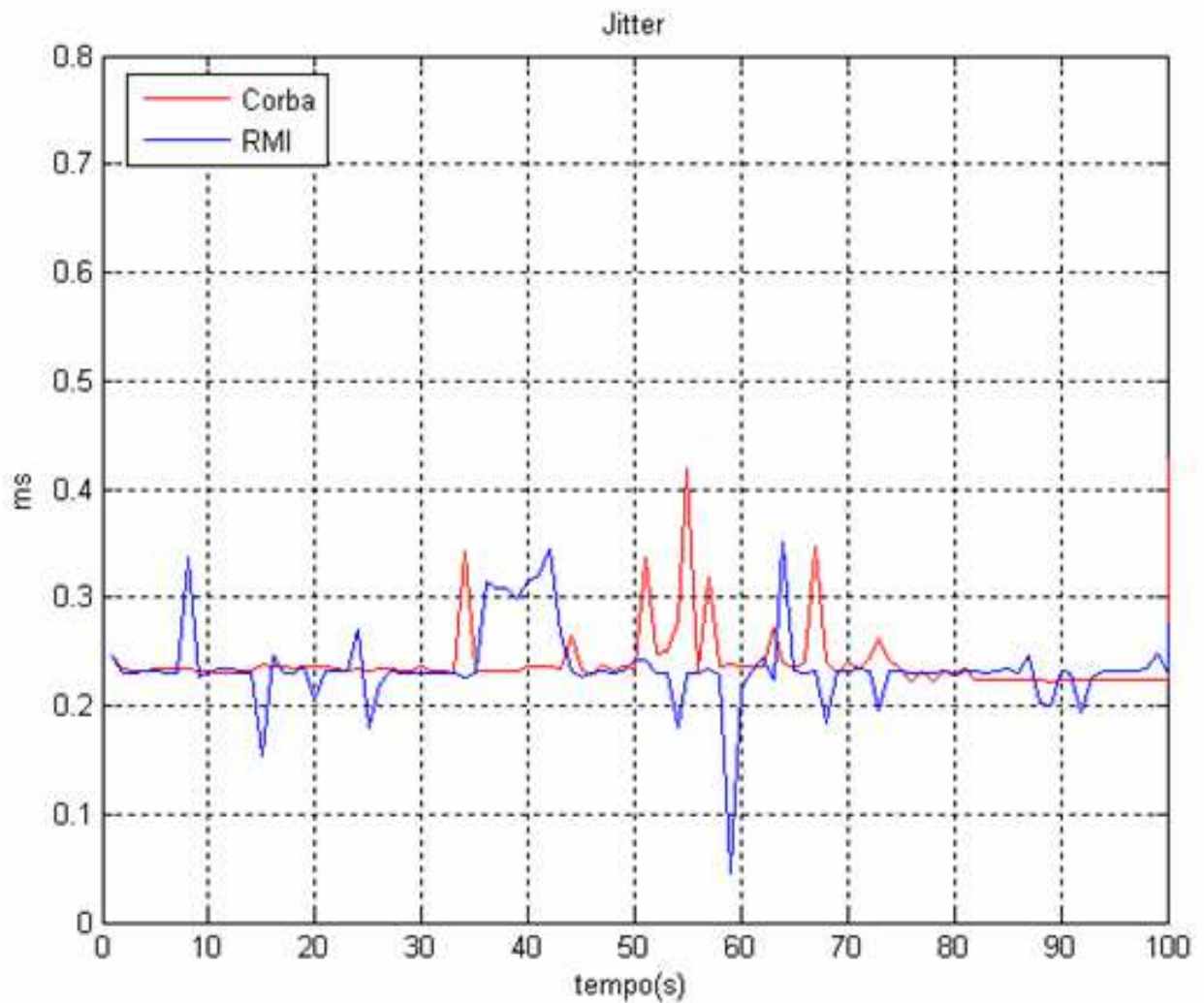


Figura 4.6: Gráfico comparativo do jitter sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

Nesta taxa de transmissão, de acordo com o gráfico, é observa uma variação de atraso mais uniforme para as duas tecnologias. Assim como nas taxas de transmissão anteriores,

as tecnologias apresentaram um jitter mais elevado durante a transmissão. Sendo que RMI possui uma variação de atraso constante enquanto CORBA apresenta picos mais elevados, mostrando que consome os recursos da rede somente enquanto estabelece uma comunicação direta entre cliente e servidor, como já foi explicado anteriormente.

4.7.1.3.4 Taxa - 40 Mb

Na figura 4.7 temos o gráfico comparativo do jitter para a taxa de transmissão 40Mb.

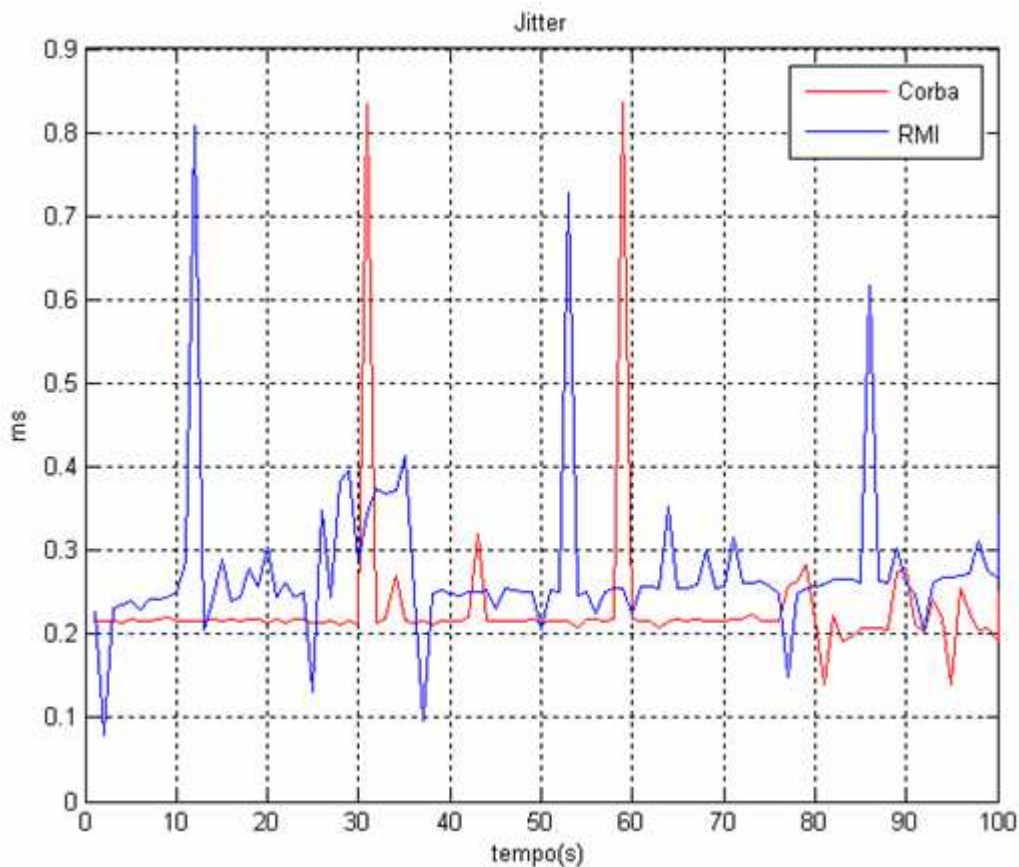


Figura 4.7: Gráfico comparativo do jitter sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

Neste gráfico se observa vários picos no jitter de ambas as tecnologias, porém é notável que nesta taxa de transmissão RMI obteve um menor desempenho em relação a CORBA, haja vista que a variação de atraso durante sua operação na rede apresentou valores mais elevados.

4.7.1.3.5 Taxa - 50 Mb

Na figura 4.8 temos o gráfico comparativo do jitter para a taxa de transmissão 50Mb.

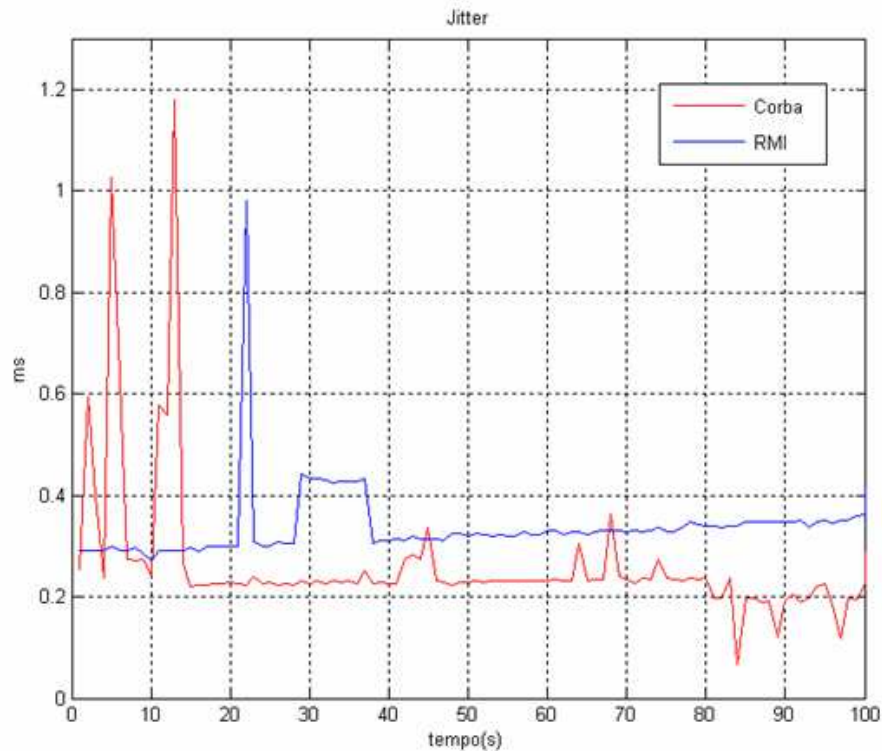


Figura 4.8: Gráfico comparativo do jitter sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

Neste gráfico se observa as condições iniciais da transmissão entre as duas estações, que enfatizam as características reais do sistema. Assim, como nas demais taxas mostradas anteriormente, o jitter apresentou uma elevação durante a inserção dos *middlewares* na rede. Como explicado antes, o jitter para RMI teve um acréscimo constante durante toda sua atuação na rede e para CORBA, somente na comunicação entre cliente e servidor propriamente dita. O gráfico denota ainda, que mesmo após o encerramento da transmissão entre servidor e cliente, o jitter para a solução RMI ainda permanece mais elevado que para CORBA, mostrando um melhor desempenho de CORBA nesta taxa de transmissão.

4.7.1.4 Perda de Pacotes

Esta métrica permite realizar um diagnóstico de quantos pacotes foram perdidos na transmissão entre a estação 1 e 2. Ela procura verificar os impactos da inserção das soluções de middleware no processo de transmissão entre outros elementos da rede. Assim como na monitoração do jitter, foram utilizadas cinco taxas de transmissão diferentes que vão de 10Mb a 50Mb. Os gráficos referentes às análises comparativas são mostrados nas figuras 4.4, 4.5, 4.6, 4.7, 4.8, respectivamente.

4.7.1.4.1 Taxa - 10 Mb

Na figura 4.9 temos o gráfico comparativo da perda de pacotes sem/durante a inserção das soluções middleware na rede, para a taxa de transmissão 10Mb.

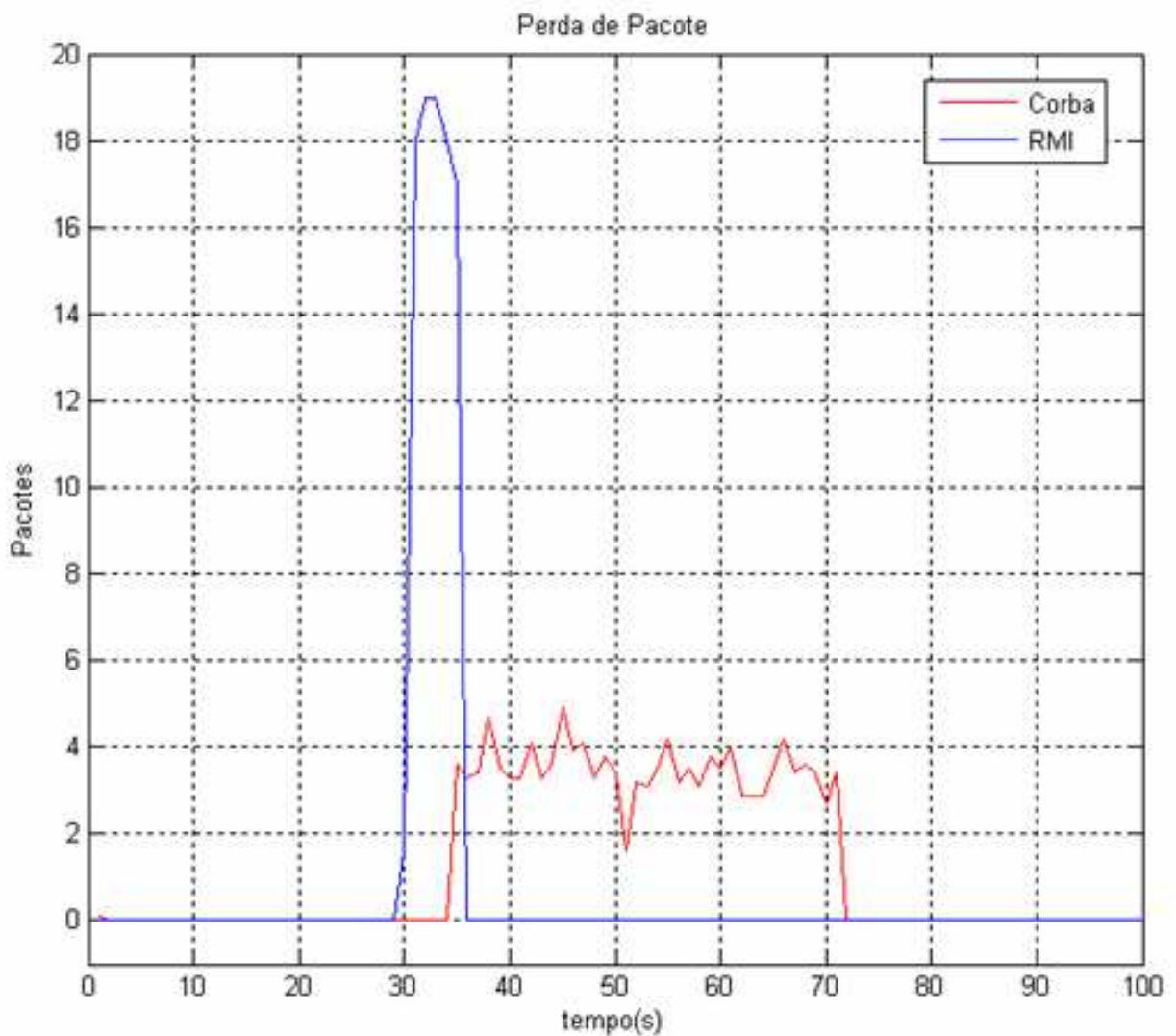


Figura 4.9: Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

4.7.1.4.2 Taxa - 20 Mb

Na figura 4.10 temos o gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware* na rede, para a taxa de transmissão 20Mb.

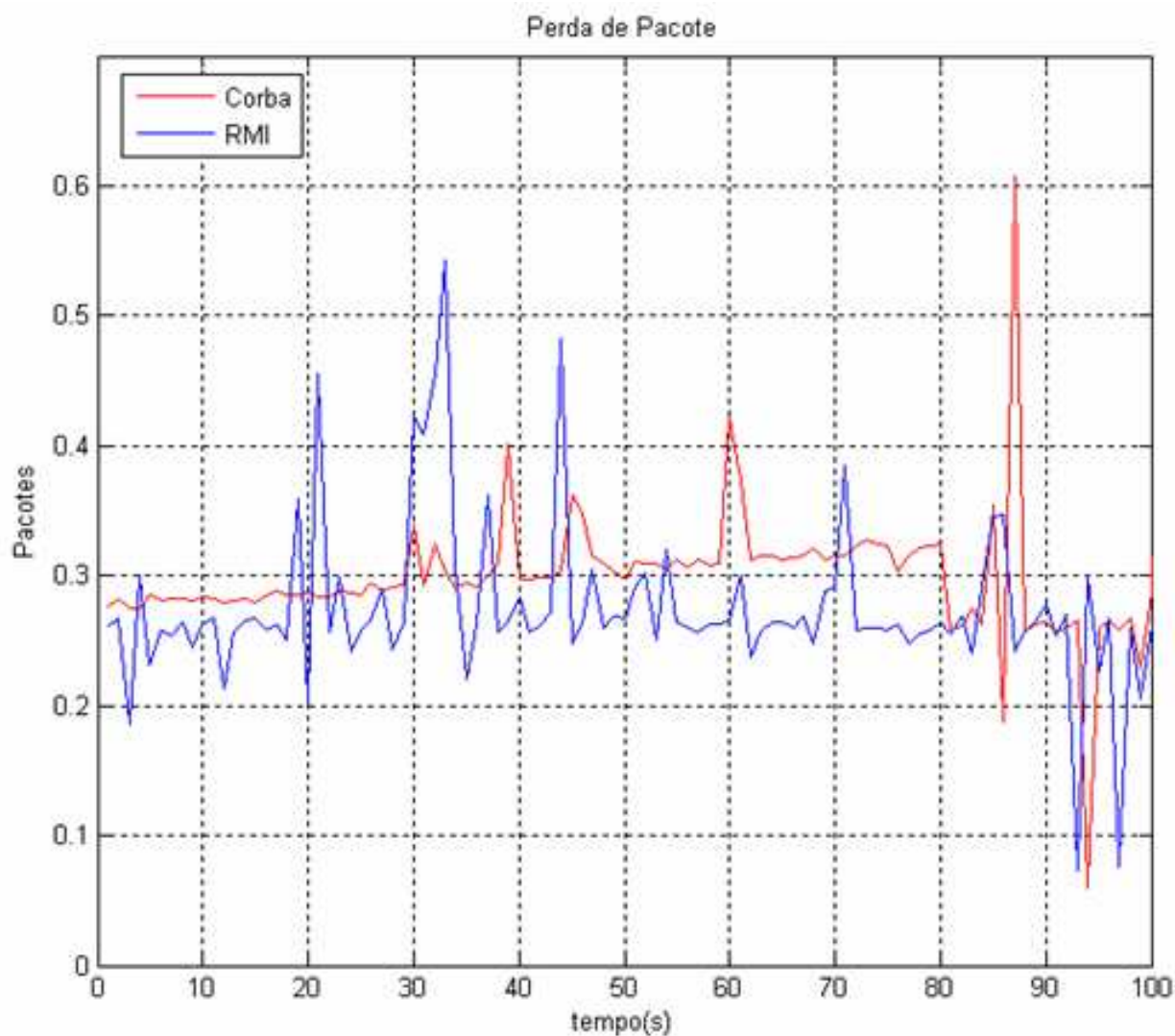


Figura 4.10: Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

4.7.1.4.3 Taxa - 30 Mb

Na figura 4.11 temos o gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware* na rede, para a taxa de transmissão 30Mb.

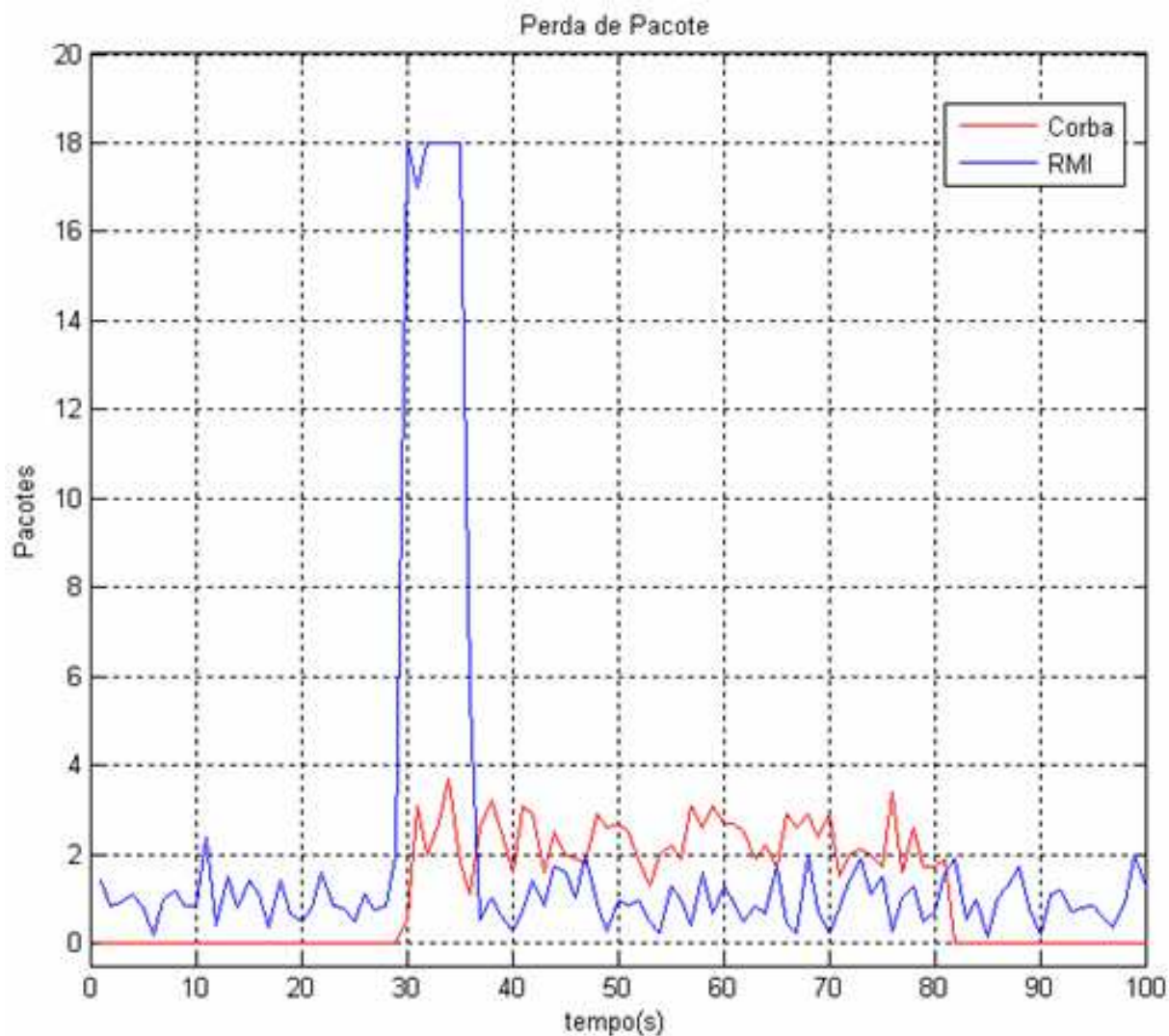


Figura 4.11: Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

4.7.1.4.4 Taxa - 40 Mb

Na figura 4.12 temos o gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware* na rede, para a taxa de transmissão 40Mb.

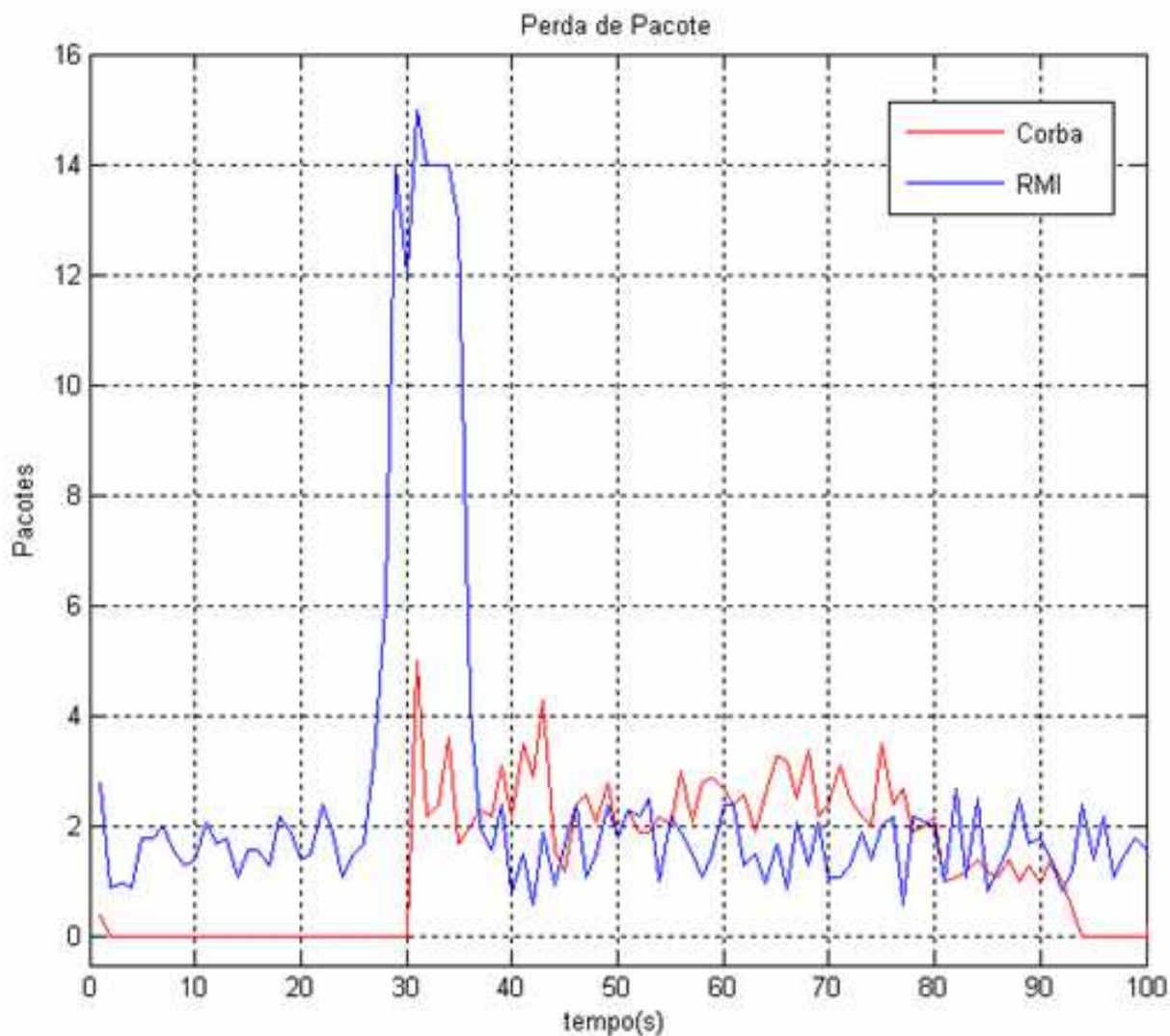


Figura 4.12: Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

4.7.1.4.5 Taxa - 50 Mb

Na figura 4.13 temos o gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware* na rede, para a taxa de transmissão 50Mb.

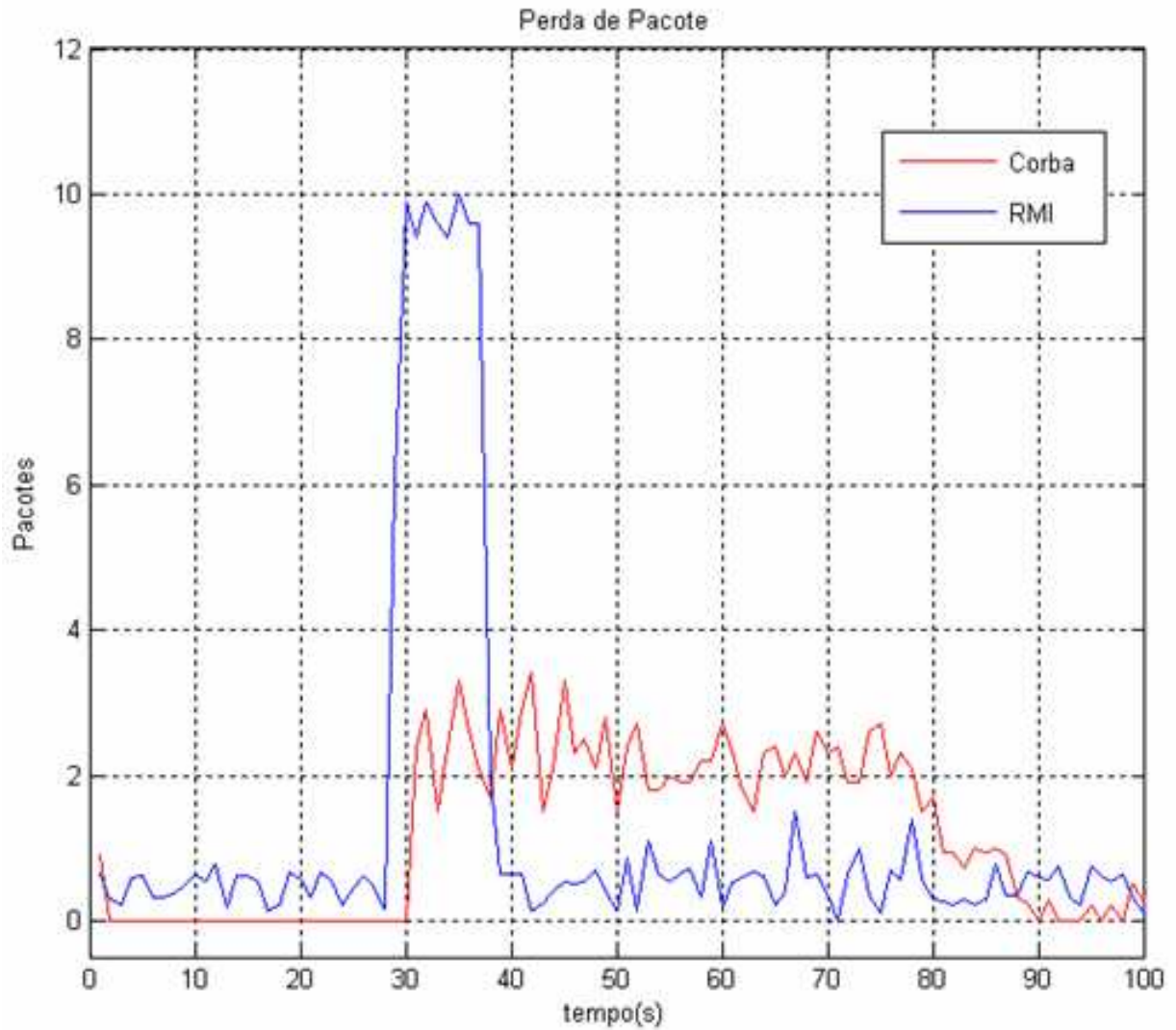


Figura 4.13: Gráfico comparativo da perda de pacotes sem/durante a inserção das soluções *middleware*

- Tempo de permanência do *middleware* RMI na rede: em média 8 segundos;
- Tempo de permanência do *middleware* CORBA na rede: em média 43 segundo.

Em todos os casos observou-se que a perda de pacote propriamente dita se deu durante a inserção do *middleware* na rede. Observa-se ainda, que RMI, apesar de permanecer um menor tempo na rede, introduziu um maior impacto no sistema. Como explicado anteriormente, isso se da pela característica da tecnologia de não se "preocupar" com os demais processos de rede enquanto atua no sistema.

4.7.2 Análise dos Recursos de Hardware

Como citado anteriormente, os recursos de *hardwares* foram analisados através do utilitário *top* do Linux. O comando *top* apresenta diversas informações sobre os processos que mais consomem recursos na máquina. Os processos podem ser ordenados por diversos critérios (uso de CPU, de memória, maior tempo de processamento utilizado, idade ou número do processo). A partir destas informações, o *top* mostra também estatísticas sobre o número de processos em execução na máquina (quantidade/estado), sobre a utilização do processador, além das informações do *uptime*, que é utilizado para mostrar a quanto tempo o sistema está em operação ininterruptamente, e também do *free* que mostra estatísticas de utilização de memória [35]. A figura 4.14 ilustra um exemplo a monitoração dos recursos de *hardware* na máquina servidor.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
7228	adriana	17	0	600m	17m	8776	S	11.6	3.5	0:00.35	java
7228	adriana	17	0	601m	51m	9.8m	S	31.5	10.2	0:01.32	java
7228	adriana	17	0	601m	99m	9.8m	S	39.3	19.8	0:02.51	java
7228	adriana	17	0	601m	166m	9.9m	S	99.8	33.1	0:05.51	java
7228	adriana	17	0	601m	247m	9.9m	S	99.3	49.1	0:08.50	java
7228	adriana	17	0	601m	336m	9.9m	S	96.7	66.7	0:11.41	java
7228	adriana	17	0	601m	336m	9.9m	S	93.8	66.7	0:14.24	java
7228	adriana	17	0	652m	364m	9.9m	S	40.3	72.2	0:15.45	java
7228	adriana	17	0	652m	349m	9.9m	S	8.0	69.2	0:15.69	java
7228	adriana	17	0	601m	298m	9.9m	S	10.3	59.3	0:16.01	java
7228	adriana	17	0	601m	298m	9.9m	S	4.0	59.3	0:16.13	java
7228	adriana	17	0	609m	329m	9.9m	S	15.2	65.2	0:16.59	java
7228	adriana	17	0	609m	364m	9.8m	S	19.3	72.3	0:17.17	java
7228	adriana	17	0	609m	368m	9.8m	S	7.2	73.1	0:17.39	java
7228	adriana	17	0	601m	379m	9768	S	29.7	75.2	0:18.29	java
7228	adriana	17	0	609m	385m	9764	S	20.0	76.4	0:18.89	java
7228	adriana	17	0	601m	379m	9764	S	18.0	75.2	0:19.43	java

Figura 4.14: Exemplo de monitoração dos recursos de hardware

Abaixo seguem as especificações de cada parâmetro mostrado no utilitário *top*.

- PID (*Process Identifier*) - mostra o número que indica o processo;
- USER - este parâmetro mostra o nome do usuário que executou o processo;
- PR e NI - mostram a prioridade e o parâmetro do comando *nice* especificado para o processo, respectivamente. O comando *nice* permite que o usuário modifique a prioridade do processo;

- VIRT, RES e SHR - estes comandos dizem respeito à utilização da memória. O primeiro corresponde ao valor de espaço total de memória utilizada pelo processo. O segundo mostra o espaço de memória física utilizada. O último corresponde a quantidade de memória compartilhada pelo processo, por exemplo, com outras bibliotecas de vínculo dinâmico.
- S - indica o status do processo:
 - R - (*Running*) é um processo executável, que está em execução;
 - S - (*Sleep*) é um processo que está suspenso, aguardando algum recurso;
 - Z - (*Zombie*) é um processo zumbi, onde é um processo que já morreu, mas que ainda está consumindo memória;
 - D - Em espera do disco, onde está aguardando uma requisição;
 - T - (*Traced*) é um processo que foi interrompido.
- %CPU - indica o valor, em porcentagem, que o processo está usando de CPU;
- %MEM: o indica o valor, em porcentagem, de quanto o processo está usando de memória;
- TIME - indica o tempo de CPU que o processo está consumindo enquanto estiver executando;
- COMMAND - mostra o nome do comando que executa aquele processo.

Foram coletadas cerca de 200 amostras, entre as transmissões TCP e UDP, a fim de observar o consumo do hardware da máquina servidor. 100 amostras correspondem à avaliação do desempenho do hardware a partir do *middleware* RMI, e as 100 restantes, do *middleware* CORBA. Ambas as tecnologias apresentaram uma mesma tendência de consumo em todas as suas amostras correspondentes, isto é, as 100 amostras coletadas para RMI mostram o mesmo consumo em todos os testes. O mesmo foi observado no *middleware* CORBA. É importante ressaltar que em todas as medições estavam ativos os mesmos processos na máquina servidor.

Na figura 4.15 mostra a demonstração do consumo do *hardware* pela solução de *middleware* RMI.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
7513	drika	16	0	600m	17m	8776	S	12.3	3.5	0:00.37	java
7513	drika	16	0	601m	31m	9.8m	S	25.0	6.2	0:01.13	java
7513	drika	16	0	601m	61m	9.8m	S	15.6	12.2	0:01.60	java
7513	drika	16	0	601m	130m	9.9m	S	65.0	25.9	0:03.56	java
7513	drika	16	0	601m	190m	9.9m	S	99.5	37.8	0:06.55	java
7513	drika	16	0	601m	286m	9.9m	S	99.5	56.9	0:09.55	java
7513	drika	16	0	601m	336m	9.9m	S	99.5	66.7	0:12.55	java
7513	drika	16	0	652m	387m	9.9m	S	90.8	76.9	0:15.28	java
7513	drika	16	0	652m	404m	7172	S	4.0	80.1	0:15.40	java
7513	drika	16	0	652m	390m	6272	S	4.0	77.4	0:15.52	java
7513	drika	16	0	601m	348m	6276	S	10.4	69.0	0:15.87	java
7513	drika	16	0	601m	348m	6276	S	4.3	69.0	0:16.00	java

Figura 4.15: Consumo dos recursos do *hardware* pelo *middleware* RMI

Na figura 4.16 mostra a demonstração do consumo do hardware pela solução de *middleware* CORBA.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8701	drika	18	0	596m	13m	6968	D	6.0	2.7	0:00.18	java
8701	drika	15	0	597m	21m	10m	S	10.6	4.2	0:00.50	java
8701	drika	15	0	598m	42m	11m	S	26.9	8.5	0:01.31	java
8701	drika	15	0	598m	91m	11m	S	29.3	18.2	0:02.20	java
8701	drika	15	0	598m	161m	11m	S	92.2	32.1	0:05.03	java
8701	drika	15	0	598m	238m	11m	S	99.1	47.3	0:08.03	java
8701	drika	15	0	598m	340m	12m	S	99.0	67.5	0:11.06	java
8701	drika	15	0	598m	340m	12m	S	95.2	67.5	0:13.93	java
8701	drika	15	0	649m	401m	11m	S	57.9	79.6	0:15.67	java
8701	drika	15	0	649m	392m	11m	S	3.0	77.8	0:15.76	java
8701	drika	15	0	598m	343m	8300	S	36.5	68.1	0:17.31	java
8701	drika	15	0	598m	353m	8300	S	93.2	70.0	0:20.11	java
8701	drika	15	0	598m	353m	8300	S	99.8	70.0	0:23.11	java
8701	drika	15	0	598m	353m	8300	S	99.8	70.0	0:26.11	java
8701	drika	15	0	598m	353m	8300	S	99.5	70.0	0:29.10	java
8701	drika	15	0	598m	353m	8300	S	99.2	70.0	0:32.08	java
8701	drika	15	0	598m	353m	8300	S	99.8	70.0	0:35.08	java
8701	drika	15	0	598m	353m	8300	S	99.2	70.0	0:38.06	java
8701	drika	15	0	598m	353m	8300	S	99.9	70.0	0:41.07	java
8701	drika	15	0	598m	353m	8300	S	99.4	70.0	0:44.06	java
8701	drika	15	0	598m	353m	8300	S	99.5	70.0	0:47.05	java
8701	drika	15	0	598m	353m	8300	S	99.8	70.0	0:50.05	java
8701	drika	15	0	598m	353m	8324	S	40.6	70.0	0:51.27	java

Figura 4.16: Consumo dos recursos do *hardware* pelo *middleware* CORBA

De acordo com as respectivas figuras, observa-se que ambas as tecnologias consomem os recursos máximos suportados pela máquina servidor durante os processos de consulta a base de dados, serialização dos dados e envio a máquina cliente. É notável que RMI possui um tempo de permanência na rede bem menor que CORBA e por conseguinte consome por menos tempo os recursos oferecidos pela máquina servidor.

4.8 CONSIDERAÇÕES FINAIS

De posse de todas as principais informações sobre os impactos sofridos pelo sistema a partir da inserção das tecnologias de *middleware* discutidas ao longo do trabalho, é possível realizar um diagnóstico sobre o comportamento do sistema distribuído, isto é, pode-se observar o ponto-chave na implementação de *middleware*: transparência x eficiência.

Antes de se adotar qualquer das duas soluções é de suma importância que características como escalabilidade, tolerância a falhas, transparência e confiabilidade

sejam consideradas, isto é, a solução de *middleware* deve ser adequada ao propósito geral do ambiente a ser aplicado.

Um dos fatores que mais viabilizaria a implementação de quaisquer das duas soluções consiste na menor interferência no tráfego da rede e na utilização de recursos de CPU e memória. Este parâmetro foi observado a partir da avaliação do desempenho do sistema pela técnica de aferição de dados, que se utilizou de valores reais para determinar o comportamento do ambiente a partir da utilização das duas soluções de *middleware*.

Em todos os casos, a tecnologia RMI apresentou um pior gerenciamento dos recursos da rede, isto é, enquanto esteve em operação utilizou a maior parte dos recursos disponíveis, como a largura da banda, por exemplo, prejudicando assim os demais processos da rede. CORBA se mostrou mais eficiente neste quesito e apesar de permanecer por mais tempo na rede não utilizou o máximo de seus recursos, caracterizando um diferencial em relação à RMI.

Destaca-se também que RMI obteve melhor desempenho em relação ao consumo dos recursos de hardware, pois permaneceu menos tempo na rede, consumindo por menos tempo recursos como uso da CPU e memória física. CORBA por sua vez consumiu mais estes recursos haja vista que permaneceu na rede por mais tempo.

5 CONCLUSÕES

5.1 CONTRIBUIÇÕES DO TRABALHO

Este trabalho apresentou os conceitos e características de soluções de interoperabilidade. Além disso, apresentou ainda, um estudo comparativo entre as duas tecnologias de objetos distribuídos largamente utilizados atualmente - RMI e CORBA. Suas principais características foram descritas, bem como sua infra-estrutura e organização dentro da plataforma de objetos distribuídos.

Em seguida foram apresentadas comparações de desempenho entre as duas tecnologias utilizando a técnica de aferição de dados. Esta técnica utilizou dados reais extraídos do ambiente heterogêneo que possibilitaram a realização de um diagnóstico a cerca da utilização de cada uma das soluções, levando em consideração as mesmas condições de *hardware* e *software* do ambiente em cada teste.

Com isso, foi possível investigar parâmetros importantes em uma rede de trabalho, tais como: vazão, variação do atraso, consumo da banda e perda de pacotes, durante a inserção das duas soluções de middlewares abordadas, a fim de se obter um conhecimento a priori da implementação destas soluções de *middleware*.

Com isso, obteve-se que a escolha da tecnologia ficaria a cargo mais detalhados como: complexidade dos sistemas heterogêneos, sistemas operacionais, protocolos, entre outros mais específicos.

Dentre as contribuições deste trabalho estão:

- Avaliação comparativa do desempenho do sistema real a partir da inserção de soluções de *middleware* para prover a interoperabilidade;
- Investigação da viabilidade da implementação dessas soluções para um ambiente semelhante ao utilizado para o estudo de caso;
- Observação comparativa da implementação de transparência, levando em consideração a eficiência do sistema;

- Manipulação de técnica de aferição de dados, importante técnica de avaliação de desempenho, para estabelecer um diagnóstico a cerca do sistema a partir da carga gerada pelas soluções *middleware* utilizadas;
- Divulgação futura do trabalho (ou parte dele) em forma de artigo científico à comunidade especializada.

5.2 DIFICULDADES ENCONTRADAS

No que diz respeito às dificuldades encontradas no decorrer do desenvolvimento deste trabalho, as mesmas advêm do conhecimento e manipulação da tecnologia CORBA, isto é, na escolha da melhor distribuição para a implementação de acordo com o ambiente em estudo. Além disso, o conhecimento e mapeamento do padrão IDL, utilizado para definir as interfaces do servidor CORBA, não é uma tarefa trivial, o que requer tempo para o entendimento e manipulação.

A aferição por meio da coleta de dados extraídos de um sistema real por meio de softwares que monitoram o comportamento do sistema, consistiu em um processo demorado e em muitos casos, vários testes precisaram ser refeitos a fim de se obter uma convergência nas informações. Os dados foram colhidos de forma não automática e foram interpretados individualmente, gerando um grande atraso no estabelecimento do diagnóstico a respeito do sistema.

O *software* utilizado para a coleta dos dados gerava vários arquivos de *trace* individuais os quais necessitaram de um tratamento prévio, ou seja, adequação ao formato correto, para que pudessem ser manipulados pelo programa gerador dos gráficos comparativos. Isto requereu também uma grande parcela de tempo, contribuindo para o atraso no estabelecimento do diagnóstico do sistema.

Além disso, o *software* utilizado para monitorar o desempenho da rede, mostrou algumas limitações como: a máxima transmissão da rede, que assumiu valores em torno de 50Mb, impossibilitando assim a monitoração da capacidade total de transmissão do sistema; e ainda, através dele, não foi possível a avaliação de um importante parâmetro observado na avaliação de desempenho de sistemas, a probabilidade de bloqueio.

5.3 TRABALHOS FUTUROS

Como trabalhos futuros, temos:

- Avaliação comparativa do desempenho do sistema em diferentes cenários, ou seja, cliente e servidor na mesma máquina, clientes e servidor em máquinas diferentes e o servidor sendo acessado por vários clientes ao mesmo tempo. Assim poderiam ser avaliados parâmetros importantes como: probabilidade de bloqueio e tempo de atendimento, onde seriam envolvidos conceitos relacionados a sistemas de filas;
- Outra sugestão seria a implementação de interoperabilidade entre diversos elementos operando em redes distintas, isto é, com protocolos de comunicação e velocidades de tráficos diferentes. Neste cenário poderia ser feito o estudo comparativo entre as duas tecnologias de *middleware* apresentadas neste trabalho a fim de investigar qual tecnologia apresentaria o melhor desempenho;
- Por fim, poderia ser incluído como um possível trabalho, um estudo comparativo em um importante cenário heterogêneo real, isto é, a comunicação entre cliente e servidor separados por *firewalls* na Internet.

Referências Bibliográficas

- [1] MAHMOUND, QUSAY H.; **Middleware for Communications**. John Wiley Sons, Ltd, 2004.
- [2] OLIVEIRA, E. L. **Interoperabilidade de Sistemas Legados: Alternativas de Implementações e Planejamento por Simulação**. 2005. 84f. Dissertação (Mestrado em Engenharia Elétrica) - Centro Tecnológico, Universidade Federal do Pará, Belém, 2005.
- [3] COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; **Distributed Systems - Concepts and Designs**. Addison - Wesley, 3º ed. 2001.
- [4] VINOSKI, E.; Where is middleware. IEEE Internet Computing, <http://computer.org/internet/>, march - april, 2002.
- [5] PAEPCKE, A.; CHANG, C.; K., MOLINA, H. G.; WINOGRAD, T.; Interoperability for Digital Libraries Worldwide. Communications of the ACM, April 1998/Vol. 41, No.4.
- [6] PAULOVICH, F. V. Middlewares em sistemas distribuídos (Notas de didáticas). 2002. (Desenvolvimento de material didático ou instrucional - Universidade Federal de São Carlos - Departamento de Computação).
- [7] CALABREZ, C. E. Uma Comparação entre Diversas Tecnologias de Comunicação de Objetos Distribuídos em Java. 2004. 110f. Dissertação (Mestrado em Engenharia de Computação). Instituto de Computação, Universidade de Campinas, Campinas, 2004.
- [8] Disponível em: <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>. Acessado em: 21 jan. 2007.
- [9] Dynamic code downloading using RMI (Using the java.rmi.server.codebase Property). Disponível em: <http://java.sun.com/j2se/1.4.2/docs/guide/rmi/codebase.htm>. Acessado em: 17 jan. 2007.

- [10] Remote Object Activation - Tutorials - Disponível em: <http://java.sun.com/j2se/1.4.2/docs/guide/rmi/activation.html>. Acessado em: 19 nov. 2006
- [11] CORBA: Integrating diverse applications within distributed heterogeneous environments - Steve Vinoski - IEEE Communications Magazine, Volume 35, Issue 2 (Feb. 1997).
- [12] New Features for CORBA 3.0 - Steve Vinoski - Communications of the ACM, Volume 41, Issue 10 (Oct. 1998)
- [13] Distributed Programming with Java, Chapter 11: Overview of CORBA - Qusay H. Mahmoud - January 2001 - <http://developer.java.sun.com/developer/Books/corba/>. Acessado em: 19 nov. 2006.
- [14] Introduction to Corba. Disponível em: <http://java.sun.com/developer/onlineTraining/corba/cor>. Acessado em: 19 nov. 2006.
- [15] OMG - Object Management Group. Disponível em: <http://www.omg.org> . Acessado em: 20 nov. 2006.
- [16] IONA Products - Orbix. Disponível em: <http://www.iona.com/products/orbix.htm>. Acessado em 26 nov. 2006.
- [17] Borland Enterprise Server, VisiBroker Edition. Disponível em: <http://www.borland.com/besvisibroker>. Acessado em: 26 nov. 2006.
- [18] Java IDL. Disponível em: <http://java.sun.com/products/jdk/idl>. Acessado em: 19 nov. 2006.
- [19] JacORB. Disponível em: <http://www.jacorb.org>. Acessado em: 26 nov. 2006.
- [20] History of CORBA. Disponível em: http://www.omg.org/gettingstarted/history_of_corba.htm. Acessado em: 19 nov. 2006.
- [21] OMG Security. Disponível em: http://www.omg.org/technology/documents/formal/omg_security. Acessado em: 21 jan. 2007.

- [22] Boniati, B. B.; Olson, A. Q.; Padoin, E.L. Interoperabilidade Em Sistemas Utilizando Web Services Como Middlewares. In II Simpósio de Informática da Região Centro / RS - Santa Maria. Santa Maria, RS. Agosto/2003
- [23] SCHNEIDER, F.R.; Integração de Aplicações Empresariais Utilizando Web Services. Dissertação de mestrado. Universidade Federal de Santa Catarina, Florianópolis, 2003.
- [24] BERTINO, E.; FERRARI, E.; XML and Data Integration. IEEE Internet Computing, <http://computer.org/internet/>, november - december, 2001.
- [25] GOLDCHLEGER, A. InteGrade: Um Sistema de Middleware para Computação em Grade Oportunista. 2004. 124f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2004.
- [26] MAHAJAN, S.; CHEN, J.; CORBA on WWW: evaluative framework for interoperability issues. Technology of Object-Oriented Languages, 1998. TOOLS 27. Proceedings 22-25 Sept. 1998 Page(s):351 - 360
- [27] BALBINOT, R.; SILVEIRA, J.G.; NETO, J.A.O.; CASTELLO, F.C.; VIEIRA, A.R.; QUADRA, A.; Interoperability of mobile location systems. Communications, Computers and signal Processing, 2003. PACRIM. 2003 IEEE Pacific Rim Conference on volume 2, 28-30 Aug. 2003 Page(s):887 - 890 vol.2
- [28] DELICATO, F.; PIRES, P.; LAGES, A.; REZENDE, J.; PIRMEZ, L. Middleware Orientado a Serviços para Redes de Sensores sem Fio. In XXII Simpósio Brasileiro de Redes de Computadores (SBRC 2004). Gramado, RS. Maio/2004.
- [29] Comparison of CORBA and Java RMI Base on Performance Analysis. Disponível em: <http://citeseer.ist.psu.edu/juric98comparison.html>. Acessado em 14 jan. 2007
- [30] Estudo comparativo detalhado do desempenho de Java/CORBA, Java/RMI. Java/ICE e Java/SOAP - Disponível em: <http://gsd.ime.usp.br/kon/MAC5755/projetos/Emilio/RelatorioFinal/>. Acessado em 14 jan. 2007.
- [31] DEITEL, H. M. et al, 2003. XML: Como Programar. Bookman, 2003.

-
- [32] RAJ JAIN. The Art of Computer Systems. Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons, 1991.
- [33] Disponível em: <http://www.gta.ufrj.br/~mdavid/xml.htm>. Acessado em 29 jan. 2007.
- [34] Disponível em: http://java.sun.com/developer/technicalArticles/RMI/rmi_corba/. Acessado em: 02 jan. 2007.
- [35] Disponível em: http://www.4newbies.com.br/arts_view.php?id=115. Acessado em : 21 jan. 2007.

A Códigos Fontes

SOLUÇÕES *MIDDLEWARE* RMI E CORBA

Aqui serão apresentados os códigos-fonte dos programas utilizados neste trabalho. Os programas denotam os algoritmos das soluções de *middleware* RMI e CORBA, os quais fazem todo o monitoramento, extração de informações de banco de dados para arquivos XML e transferência dos arquivos para um cliente que requisita o serviço.

*Middleware para prover interoperabilidade em um ambiente
cliente/servidor - Soluções utilizando RMI e CORBA.*

**CLASSES UTILIZADAS PARA CONEXÃO COM O BANCO DE
DADOS E CONVERSÃO DOS DADOS EM ARQUIVO NO FORMATO
XML.**

a) Classe Conexão

```
/*
** Conexao.java
** Classe de conexão utizando JDBC
** Desenvolvido por: Edvar Oliveira
** Universidade Federal do Pará
** Programa de Pós-Graduação em Engenharia Elétrica
** Laboratório de Computação Aplicada
*/

//package middrmi;

import java.sql.*;

import java.util.Vector;

public class Conexao

private Connection c;
```

```
private ResultSet table;

public Conexao() throws ClassNotFoundException

// carrega driver

try Class.forName(Constants.DRIVERNAME);

catch(ClassNotFoundException e1)

System.out.println( "Erro na conexao1:No Driver ");

public void connect(String user, String password) throws SQLException

// estabelece conexao com banco de dados

c = DriverManager.getConnection("jdbc:mysql://fourier:3306/"

+ Constants.BANCO + "?user="+ user + "&password="+ password);

public void connect(String user) throws SQLException

connect(user, "root");

public void connect() throws SQLException, ClassNotFoundException

try

// estabelece conexao com banco de dados

if (Constants.MARCADOBANCO.equals("mysql"))

c = DriverManager.getConnection("jdbc:mysql://fourier:3306/"

+ Constants.BANCO + "?user="+

Constants.USER

+ "password="+ Constants.PASSWORD);

System.out.println("Conectado ao MySql.");

if (Constants.MARCADOBANCO.equals("firebird"))

c = DriverManager.getConnection(

"jdbc:firebirdsql:localhost/3050:"

+ Constants.PATHBANCO,

Constants.USER,
```

```
        Constantes.PASSWORD);

        System.out.println("aqui");

        catch (SQLException e)

        System.out.println("Problemas na conexao com a fonte de dados,verifique o

        SQL passado");

        public void query(String sql) throws SQLException

        try

        Statement s = c.createStatement();

        table = s.executeQuery(sql);

        catch(SQLException e)

        System.out.println("Problemas na conexao com a fonte de dados,verifique o

        SQL passado");

        public ResultSet getTable()

        return table;

        public Vector structure() throws SQLException

        // obtem e transforma em String a estrutura da tabela

        Vector str = new Vector();

        ResultSetMetaData m = table.getMetaData();

        int colCount = m.getColumnCount();

        for (int i = 1;i <= colCount; ++i)

        str.addElement(m.getColumnName(i));

        return str;

        // trecho -> confere quantas posições de registros possui o resultset

        int x = 0;

        table.first();

        while (table.next())

        x++;
```

```
table.first();

// matriz onde será armazenada todas os registros da tabela
String[] tupla = new String[x][colCount];

int i = 0;

while (table.next())

// flag para verificar será primeira posição do Resultset
if (i == 0)

table.first();

try

for (int ky = 0; ky < colCount; ky++)

tupla[i][ky] = table.getString(ky + 1); //problema esta aqui, nao retira a
informacao de table e coloca na matriz de Strings tupla.

catch(Exception e)e.printStackTrace();

i++;

// instancio um objeto registro para saber algumas características da
Registro r = new Registro(tupla, x, colCount);

return r;

public Vector showTables() throws SQLException

int i = 1;

// obtem e transforma em String o conteudo da tabela

int colCount = table.getMetaData().getColumnCount();

Vector tabName = new Vector();

while (table.next())

tabName.addElement(table.getString(i));

System.out.println("table.getString("+i+")="+table.getString(i));

return tabName;
```

```
public void close() throws SQLException
```

```
c.close();
```

b) Classe Abstrata Constante

```
/*
```

```
** Constante.java
```

```
** Classe abstrata para setar alguns parametros de conexão
```

```
** com o banco de dados
```

```
** Desenvolvido por: Edvar Oliveira
```

```
** Universidade Federal do Pará
```

```
** Programa de Pós-Graduação em Engenharia Elétrica
```

```
** Laboratório de Computação Aplicada
```

```
*/
```

```
//package middrmi;
```

```
public abstract class Constantes
```

```
/******
```

```
** Atributos de Usuario e Senha para conexao com o
```

```
** banco de dados
```

```
*****/
```

```
public static final String USER = "root";
```

```
public static final String PASSWORD = ;
```

```
public static final String BANCO = "metricas";
```

```
/******
```

```
** Local onde ira ser armazenado o arquivo XML
```

```
** temporario
```

```
*****/
```

```
public static final String LOCALARMAZENAMENTO="C:
```

temp

”;

```

/*****

** setando o driver JDBC de acordo com o banco a ser

** conectado

*****/

public static final String MARCADOBANCO="mysql";

public static final String DRIVERNAME = "com.mysql.jdbc.Driver";

public static final String sql = "show tables";

```

c) Classe de Geração de Arquivo XML a partir de uma conexão com o banco de dados

```

/*

** GeraBdXml.java

** Aplicação para leitura do conteúdo de banco de dados e

** criação de documento XML

** Utiliza: Xerces - DOM

** Universidade Federal do Pará

** Programa de Pós-Graduação em Engenharia Elétrica

** Laboratório de Computação Aplicada

*/

//package middrmi;

import java.io.*;

import java.util.Vector;

import org.w3c.dom.*;

import javax.xml.parsers.*;

// xerces

import org.apache.xml.serialize.*;

public class GeraBdXml

```

```
private Document document;

private FileOutputStream fout;

private DataOutputStream out;

private File nomearquivo1;

private String arquivo = null;

private Element root;

private Tabela tab;

private MinnerTable mn;

public GeraBdXml()

public Node createContactNode()

Element tabelageral = document.createElement("Tabelas");

mn = new MinnerTable();

// nome das tabelas do banco

Vector ntab = mn.quantTabelas();

for (int k = 0; k < ntab.size(); k++)

tab = mn.minera((String) ntab.elementAt(k));

//cria um elemento campo

Element campo = document.createElement("campo");

// cria um elemento tabela

Element tabela = document.createElement("tabela");

// crio um atributo e anexo ao elemento tabela

Attr nometabela = document.createAttribute("nome");

nometabela.setValue((String) ntab.elementAt(k));

tabela.setAttributeNode(nometabela);

// informacoes da tabela

Vector campos = tab.getCampos();
```

```
String[][] reg = tab.getRegistro();

// crio um atributo e anexo ao elemento tabela
Attr quantidade = document.createAttribute("qntd");
quantidade.setValue(Integer.toString(tab.getLinha()));
tabela.setAttributeNode(quantidade);

// loops para gerar os nos tabela
for (int j = 0; j < tab.getLinha(); j++)
for (int i = 0; i < tab.getColuna(); i++)

// cria o primeiro e ultimo elemento
String buf = (String) campos.elementAt(i);
Element registro = document.createElement(buf);

// anexo um valor ao elemento musica
registro.appendChild(document.createTextNode(reg[j][i]));

// anexo os filhos e atributos ao elemento diretorio campo.appendChild(registro);
tabela.appendChild(campo);

campo = document.createElement("campo");
tabelageral.appendChild(tabela);

return tabelageral;

// Salva arquivo xml

public String salvar()
try
String cf;

if (Constantes.LOCALARMAZENAMENTO.length() > 0)
cf = Constantes.LOCALARMAZENAMENTO + Constantes.BANCO
+ ".xml";
else
```



```
cf = "bancodedados.xml";

OutputFormat format = new OutputFormat(document);

OutputStream fileOut = new FileOutputStream(cf);

XMLSerializer serial = new XMLSerializer(fileOut, format);

serial.serialize(document);

return cf;

catch (IOException e)

System.err.println(e.getMessage());

return null;

// metodo q le os diretorio e sub-diretórios de um determinado File e passa

// como

// parametro o conteudo e path para a criação de um documento XML

public String listaT()

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

try

// get DocumentBuilder

DocumentBuilder builder = factory.newDocumentBuilder();

// cria um no raiz (referencia)

document = builder.newDocument();

catch (ParserConfigurationException pce)

pce.printStackTrace();

root = document.createElement("bancodedados");

document.appendChild(root);

// adiciona um comentário a um documento XML

Comment simpleComment = document

.createComment("Criacao de documentos XML ");
```

```
root.appendChild(simpleComment);

Node tabNode = createContactNode();

root.appendChild(tabNode);

mn.finaliza();

String fname = salvar();

return fname;
```

d) Classe principal para gerar o arquivo XML temporário

```
/*

** MinnerTable.java

** Classe principal para gerar o arquivo xml temporario

** Universidade Federal do Para

** Programa de Pos-Graduação em Engenharia Eletrica

** Laboratorio de Computação Aplicada

*/

//package middrmi;

import java.sql.SQLException;

import java.util.Vector;

public class MinnerTable

private static Vector tableN;

private static Vector tableNN;

private static Vector structN;

private static Registro tuplaN;

private Conexao db;

public MinnerTable()

try

db = new Conexao();
```

```
db.connect();

catch (SQLException E)

System.err.println("SQLException: " + E.getMessage());

System.err.println("SQLState: " + E.getSQLState());

System.err.println("VendorError: " + E.getErrorCode());

catch (Exception e)

e.printStackTrace();

public Vector quantTabelas()

try

// mostra todas as tabelas do bd

db.query(Constants.sql);

tableNN = db.showTables();

catch (SQLException e)

// TODO Auto-generated catch block

e.printStackTrace();

return tableNN;

public Tabela minera(String buf)

try

// consulta

db.query("Select * from " + buf + "where centro<400000");

// estrutura da tabela escolhida

structN = db.structure();

// retorna um objeto registro

tuplaN = db.showAll();

catch (SQLException E)

System.err.println("SQLException: " + E.getMessage());
```

```
System.err.println("SQLState: " + E.getSQLState());

System.err.println("VendorError: " + E.getErrorCode());

catch (Exception e)

e.printStackTrace();

String[][] vet = tuplaN.getRegistro();

Tabela tab = new Tabela(structN, vet, tuplaN

.getColuna(), tuplaN.getLinha(), Constantes.BANCO);

return tab;

public void finaliza()

try

// fecha conexao com banco de dados

db.close();

catch (SQLException e)

// TODO Auto-generated catch block

e.printStackTrace();
```

e) Classe auxiliar de atributos de registros

```
/*

** Registro.java

** Classe de atributos auxiliares

** Universidade Federal do Pará

** Programa de Pós-Graduação em Engenharia Elétrica

** Laboratório de Computação Aplicada

*/

//package middrmi;

public class Registro

private String[][] registro;
```

```
private int coluna;

private int linha;

public Registro(String[][] registro, int linha, int coluna)

this.registro = registro;

this.coluna = coluna;

this.linha = linha;

/**
 * @return Returns the registro.
 */
public String[][] getRegistro()

return registro;

/**
 * @param registro The registro to set.
 */
public void setRegistro(String[][] registro)

this.registro = registro;

/**
 * @return Returns the coluna.
 */
public int getColuna()

return coluna;

/**
 * @param coluna The coluna to set.
 */
public void setColuna(int coluna)

this.coluna = coluna;
```

```
/**
 * @return Returns the linha.
 */
public int getLinha()
return linha;

/**
 * @param linha The linha to set.
 */
public void setLinha(int linha)
this.linha = linha;
```

f) Classe auxiliar de atributos de tabelas

```
/*
** Tabela.java
** Classe de atributos auxiliares
** Universidade Federal do Pará
** Programa de Pós-Graduação em Engenharia Elétrica
** Laboratório de Computação Aplicada
*/

//package middrmi;

import java.util.Vector;

public class Tabela

private Vector campos;

private String[][] registro;

private int coluna;

private int linha;

private String banco;
```

```
public Tabela( Vector campos, String[][] registro,
int coluna, int linha, String banco)

this.campos = campos;

this.coluna = coluna;

this.linha = linha;

this.registro = registro;

this.banco = banco;

/**
 * @return Returns the campos.
 */
public Vector getCampos()

return campos;

/**
 * @param campos
 * The campos to set.
 */
public void setCampos(Vector campos)

this.campos = campos;

/**
 * @return Returns the coluna.
 */
public int getColuna()

return coluna;

/**
 * @param coluna
 * The coluna to set.
```

```
*/

public void setColuna(int coluna)

this.coluna = coluna;

/**

* @return Returns the linha.

*/

public int getLinha()

return linha;

/**

* @param linha

* The linha to set.

*/

public void setLinha(int linha)

this.linha = linha;

/**

* @return Returns the registro.

*/

public String[][] getRegistro()

return registro;

/**

* @param registro

* The registro to set.

*/

public void setRegistro(String[][] registro)

this.registro = registro;

/**
```



```
* @return Returns the banco.  
  
*/  
  
public String getBanco()  
  
return banco;  
  
/**  
  
* @param banco The banco to set.  
  
*/  
  
public void setBanco(String banco)  
  
this.banco = banco;
```

CLASSES DE IMPLEMENTAÇÃO DA SOLUÇÃO RMI

a) Interface para definição de métodos remotos

```
/*  
  
** Interface.java  
  
** Interface para geração de código RMI para transferência  
  
** do arquivo xml serializado  
  
** Universidade Federal do Pará  
  
** Programa de Pós-Graduação em Engenharia Elétrica  
  
** Laboratório de Computação Aplicada  
  
*/  
  
//package middrmi;  
  
import java.rmi.*;  
  
public interface Interface extends Remote  
  
public String mensagem() throws java.rmi.RemoteException;  
  
public byte[] downloadFile() throws java.rmi.RemoteException;
```

b) Classe Servidor Remoto implementado em RMI

```
/*
```

```
** ServidorRemoto.java

** Classe RMI servidor com implementação da serialização,
** onde o arquivo XML será recebido do cliente RMI
** Universidade Federal do Pará
** Programa de Pós-Graduação em Engenharia Elétrica
** Laboratório de Computação Aplicada

*/

//package middrmi;

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.rmi.RMISecurityManager;

import java.rmi.server.UnicastRemoteObject;

import java.io.*;

public class ServidorRemoto extends UnicastRemoteObject implements Inter-
face

public ServidorRemoto() throws java.rmi.RemoteException

//super();

public byte[] downloadFile() throws java.rmi.RemoteException

GeraBdXml gerar = new GeraBdXml();

byte[] buf;

String fileName=;

try

fileName = gerar.listaT();

System.out.println("Transferring file " + fileName + "...");

File file = new File(fileName);

FileInputStream in = new FileInputStream(file);

buf=new byte[in.available()];
```

```
        in.read(buf);

        in.close();

        return buf;

        catch(FileNotFoundException fne)

        System.err.println("Arquivo n encontrado ");

        System.exit(1);

        catch(Exception e)

        e.printStackTrace();

        System.out.println("Transferring file " + fileName + "...");

        return(null);

        public String mensagem() throws java.rmi.RemoteException

        // TODO Auto-generated method stub

        return "teste";

        public static void main(String[] args)

        if (System.getSecurityManager() == null)

        System.setSecurityManager(new RMISecurityManager());

        try

        System.out.println("Inicializando Servidor rodando.");

        ServidorRemoto obj = new ServidorRemoto();

        /*nomeia a instancia do objeto como MensagemServer*/

        Naming.rebind("ServidorXml", obj);

        System.out.println("Mensagem Server rodando.");

        catch (Exception e)

        System.out.println("Erro: " + e.getMessage());

        e.printStackTrace();
```

c) Classe Cliente implementada em RMI

```
/*

** SCRemoto.java

** Classe RMI cliente, onde o o arquivo XML será transferido

** para o servidor RMI

** Universidade Federal do Pará

** Programa de Pós-Graduação em Engenharia Elétrica

** Laboratório de Computação Aplicada

*/

//package middrmi;

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.io.*;

public class SCRemoto

/**

*

*/

public SCRemoto()

super();

// TODO Auto-generated constructor stub

public void getDocument()

Interface obj;

double TInicio;

double TFim;

String saida;

try

// obj = (Mensagem)Naming.lookup("//" + getCodeBase().getHost() + "/Men-
sagemServer");
```

```

    obj = (Interface)Naming.lookup("//10.57.1.113/ServidorXml");

    // tempo de inicio e o fim da transferencia do document xml

    TInicio = System.currentTimeMillis();

    byte[] stream = obj.downloadFile();

    FileOutputStream out = new FileOutputStream("arquivoT.xml");

    out.write(stream);

    out.close();

    TFim= System.currentTimeMillis();

    System.out.println("Tempo de envio + ordenação + reposta do servidor =" +
(TFim-

    TInicio)/1000);

    catch (Exception e)

    System.out.println("Erro: " + e.getMessage());

    public static void main(String[] args)

    SCRemoto scr = new SCRemoto();

    scr.getDocument();

```

CLASSES DE IMPLEMENTAÇÃO DA SOLUÇÃO CORBA

a) Interface para definição de métodos remotos

- "Interface Definition Language (IDL)

```
interface Hello
```

```

    typedef sequence<octet> Data;

    Data downloadFile();

    string sayHello();

    ;

```

- "Interface correspondente gerada a partir da compilação da IDL

```
/**
```

```

* Hello.java .

* Generated by the IDL-to-Java compiler (portable), version "3.2"

* from Hello.idl

* Quinta-feira, 18 de Janeiro de 2007 14h54min41s BRST

*/

public interface Hello extends HelloOperations, org.omg.CORBA.Object,
org.omg.CORBA.portable.IDLEntity

// interface Hello

```

- "Classes auxiliares geradas a partir da compilação da IDL

```

/**

* _HelloStub.java .

* Generated by the IDL-to-Java compiler (portable), version "3.2"

* from Hello.idl

* Quinta-feira, 18 de Janeiro de 2007 14h54min41s BRST

*/

public class _HelloStub extends org.omg.CORBA.portable.ObjectImpl imple-
ments Hello

public byte[] downloadFile ()

org.omg.CORBA.portable.InputStream $in = null;

try

org.omg.CORBA.portable.OutputStream $out = _request ("downloadFile", true);

$in = _invoke ($out);

byte $result[] = HelloPackage.DataHelper.read ($in);

return $result;

catch (org.omg.CORBA.portable.ApplicationException $ex)

$in = $ex.getInputStream ();

```

```
String _id = $ex.getId ();

throw new org.omg.CORBA.MARSHAL (_id);

catch (org.omg.CORBA.portable.RemarshalException $rm)

return downloadFile ( );

finally

_releaseReply ($in);

// downloadFile

public String sayHello ()

org.omg.CORBA.portable.InputStream $in = null;

try

org.omg.CORBA.portable.OutputStream $out = _request ("sayHello", true);

$in = _invoke ($out);

String $result = $in.read_string ();

return $result;

catch (org.omg.CORBA.portable.ApplicationException $ex)

$in = $ex.getInputStream ();

String _id = $ex.getId ();

throw new org.omg.CORBA.MARSHAL (_id);

catch (org.omg.CORBA.portable.RemarshalException $rm)

return sayHello ( );

finally

_releaseReply ($in);

// sayHello

// Type-specific CORBA::Object operations

private static String[] _ids =

"IDL:Hello:1.0"
```

```
        ;

        public String[] _ids ()

        return (String[])_ids.clone ();

        private void readObject (java.io.ObjectInputStream s) throws java.io.IOException

        String str = s.readUTF ();

        String[] args = null;

        java.util.Properties props = null;

        org.omg.CORBA.Object obj = org.omg.CORBA.ORB.init (args, props).string_to_object

(str);

        org.omg.CORBA.portable.Delegate delegate = ((org.omg.CORBA.portable.ObjectImpl)

obj)._get_delegate ();

        _set_delegate (delegate);

        private void writeObject (java.io.ObjectOutputStream s) throws java.io.IOException

        String[] args = null;

        java.util.Properties props = null;

        String str = org.omg.CORBA.ORB.init (args, props).object_to_string (this);

        s.writeUTF (str);

        // class

HelloStub

        /**

        /**

        * HelloHelper.java .

        * Generated by the IDL-to-Java compiler (portable), version "3.2"

        * from Hello.idl

        * Quinta-feira, 18 de Janeiro de 2007 14h54min41s BRST

        */

        abstract public class HelloHelper
```



```
private static String _id = "IDL:Hello:1.0";

public static void insert (org.omg.CORBA.Any a, Hello that)

org.omg.CORBA.portable.OutputStream out = a.create_output_stream ();

a.type (type ());

write (out, that);

a.read_value (out.create_input_stream (), type ());

public static Hello extract (org.omg.CORBA.Any a)

return read (a.create_input_stream ());

private static org.omg.CORBA.TypeCode __typeCode = null;

synchronized public static org.omg.CORBA.TypeCode type ()

if (__typeCode == null)

__typeCode = org.omg.CORBA.ORB.init ().create_interface_tc (HelloHelper.id

(), "Hello");

return __typeCode;

public static String id ()

return _id;

public static Hello read (org.omg.CORBA.portable.InputStream istream)

return narrow (istream.read_Object (_HelloStub.class));

public static void write (org.omg.CORBA.portable.OutputStream ostream,

Hello value)

ostream.write_Object ((org.omg.CORBA.Object) value);

public static Hello narrow (org.omg.CORBA.Object obj)

if (obj == null)

return null;

else if (obj instanceof Hello)

return (Hello)obj;

else if (!obj._is_a (id ()))
```

```
        throw new org.omg.CORBA.BAD_PARAM ();

    else

        org.omg.CORBA.portable.Delegate delegate =

        ((org.omg.CORBA.portable.ObjectImpl)obj)._get_delegate ();

        _HelloStub stub = new _HelloStub ();

        stub._set_delegate(delegate);

        return stub;

    public static Hello unchecked_narrow (org.omg.CORBA.Object obj)

    if (obj == null)

        return null;

    else if (obj instanceof Hello)

        return (Hello)obj;

    else

        org.omg.CORBA.portable.Delegate delegate =

        ((org.omg.CORBA.portable.ObjectImpl)obj)._get_delegate ();

        _HelloStub stub = new _HelloStub ();

        stub._set_delegate(delegate);

        return stub;

    /**

    * HelloHolder.java .

    * Generated by the IDL-to-Java compiler (portable), version "3.2"

    * from Hello.idl

    * Quinta-feira, 18 de Janeiro de 2007 14h54min41s BRST

    */

    public final class HelloHolder implements org.omg.CORBA.portable.Streamable

    public Hello value = null;
```

```
public HelloHolder ()

public HelloHolder (Hello initialValue)

value = initialValue;

public void _read (org.omg.CORBA.portable.InputStream i)

value = HelloHelper.read (i);

public void _write (org.omg.CORBA.portable.OutputStream o)

HelloHelper.write (o, value);

public org.omg.CORBA.TypeCode _type ()

return HelloHelper.type ();

/**
 * HelloOperations.java .
 *
 * Generated by the IDL-to-Java compiler (portable), version "3.2"
 * from Hello.idl
 *
 * Quinta-feira, 18 de Janeiro de 2007 14h54min41s BRST
 */

public interface HelloOperations

byte[] downloadFile ();

String sayHello ();

// interface HelloOperations

* HelloPOA.java .

* Generated by the IDL-to-Java compiler (portable), version "3.2"
* from Hello.idl
*
* Quinta-feira, 18 de Janeiro de 2007 14h54min41s BRST
*/

public abstract class HelloPOA extends org.omg.PortableServer.Servant

implements HelloOperations, org.omg.CORBA.portable.InvokeHandler
```

```
// Constructors

private static java.util.Hashtable _methods = new java.util.Hashtable ();

static

_methods.put ("downloadFile", new java.lang.Integer (0));

_methods.put ("sayHello", new java.lang.Integer (1));

public org.omg.CORBA.portable.OutputStream _invoke (String $method,
org.omg.CORBA.portable.InputStream in,
org.omg.CORBA.portable.ResponseHandler $rh)
org.omg.CORBA.portable.OutputStream out = null;

java.lang.Integer __method = (java.lang.Integer)_methods.get ($method);

if (__method == null)

throw new org.omg.CORBA.BAD_OPERATION (0,
org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);

switch (__method.intValue ())

case 0: // Hello/downloadFile

byte $result[] = null;

$result = this.downloadFile ();

out = $rh.createReply();

HelloPackage.DataHelper.write (out, $result);

break;

case 1: // Hello/sayHello

String $result = null;

$result = this.sayHello ();

out = $rh.createReply();

out.write_string ($result);

break;
```

```
        default:

        throw new org.omg.CORBA.BAD_OPERATION (0,

        org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);

        return out;

        // _invoke

        // Type-specific CORBA::Object operations

        private static String[] _ids =

        "IDL:Hello:1.0"

        ;

        public String[] _all_interfaces (org.omg.PortableServer.POA poa, byte[] objec-
tId)

        return (String[])_ids.clone ();

        public Hello _this()

        return HelloHelper.narrow(

        super._this_object());

        public Hello _this(org.omg.CORBA.ORB orb)

        return HelloHelper.narrow(

        super._this_object(orb));

        // class HelloPOA

        * HelloPackage/DataHelper.java .

        * Generated by the IDL-to-Java compiler (portable), version "3.2"

        * from Hello.idl

        * Quinta-feira, 18 de Janeiro de 2007 14h54min41s BRST

        */

        abstract public class DataHelper

        private static String _id = "IDL:Hello/Data:1.0";

        public static void insert (org.omg.CORBA.Any a, byte[] that)
```

```
org.omg.CORBA.portable.OutputStream out = a.create_output_stream ();

a.type (type ());

write (out, that);

a.read_value (out.create_input_stream (), type ());

public static byte[] extract (org.omg.CORBA.Any a)

return read (a.create_input_stream ());

private static org.omg.CORBA.TypeCode __typeCode = null;

synchronized public static org.omg.CORBA.TypeCode type ()

if (__typeCode == null)

__typeCode = org.omg.CORBA.ORB.init ().get_primitive_tc (org.omg.CORBA.TCKind.t

__typeCode = org.omg.CORBA.ORB.init ().create_sequence_tc (0, __typeCode);

__typeCode = org.omg.CORBA.ORB.init ().create_alias_tc (HelloPackage.DataHelper.id

(), "Data",

__typeCode);

return __typeCode;

public static String id ()

return _id;

public static byte[] read (org.omg.CORBA.portable.InputStream istream)

byte value[] = null;

int _len0 = istream.read_long ();

value = new byte[_len0];

istream.read_octet_array (value, 0, _len0);

return value;

public static void write (org.omg.CORBA.portable.OutputStream ostream,

byte[] value)

ostream.write_long (value.length);

ostream.write_octet_array (value, 0, value.length);
```

```
/**
 * HelloPackage/DataHolder.java .
 * Generated by the IDL-to-Java compiler (portable), version "3.2"
 * from Hello.idl
 * Quinta-feira, 18 de Janeiro de 2007 14h54min41s BRST
 */

public final class DataHolder implements org.omg.CORBA.portable.Streamable

public byte value[] = null;

public DataHolder ()

public DataHolder (byte[] initialValue)

value = initialValue;

public void _read (org.omg.CORBA.portable.InputStream i)

value = HelloPackage.DataHelper.read (i);

public void _write (org.omg.CORBA.portable.OutputStream o)

HelloPackage.DataHelper.write (o, value);

public org.omg.CORBA.TypeCode _type ()

return HelloPackage.DataHelper.type ();
```

b) Classe Servidor Remoto implementado em CORBA

```
// HelloServer.java

// Copyright and License

//import HelloApp.*;

import java.io.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import org.omg.PortableServer.*;
```

```
import org.omg.PortableServer.POA;

public class HelloServer

public static void main(String args[])

try

// create and initialize the ORB

ORB orb = ORB.init(args, null);

// get reference to rootpoa activate the POAManager

POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

rootpoa.the_POAManager().activate();

// create servant and register it with the ORB

HelloImpl helloImpl = new HelloImpl();

helloImpl.setORB(orb);

// get object reference from the servant

org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);

Hello href = HelloHelper.narrow(ref);

// get the root naming context

// NameService invokes the name service

org.omg.CORBA.Object objRef =

orb.resolve_initial_references("NameService");

// Use NamingContextExt which is part of the Interoperable

// Naming Service (INS) specification.

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

NameComponent path[] = ncRef.to_name( name );

ncRef.rebind(path, href);

System.out.println("HelloServer ready and waiting ...");

// wait for invocations from clients
```



```
orb.run();

catch (Exception e)

System.err.println("ERROR: " + e);

e.printStackTrace(System.out);

System.out.println("HelloServer Exiting ...");
```

c) Classe Cliente implementada em CORBA

```
//import HelloApp.*;

import java.io.*;

import java.util.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

public class HelloClient

static Hello helloImpl;

public static void main(String argv[])

try

// create and initialize the ORB

Properties props = new Properties();

props.put("org.omg.CORBA.ORBInitialHost", "fourier");

props.put("org.omg.CORBA.ORBInitialPort", "1050");

ORB orb = ORB.init(argv, props);

//ORB orb = ORB.init(args, null);

// get the root naming context

org.omg.CORBA.Object objRef =

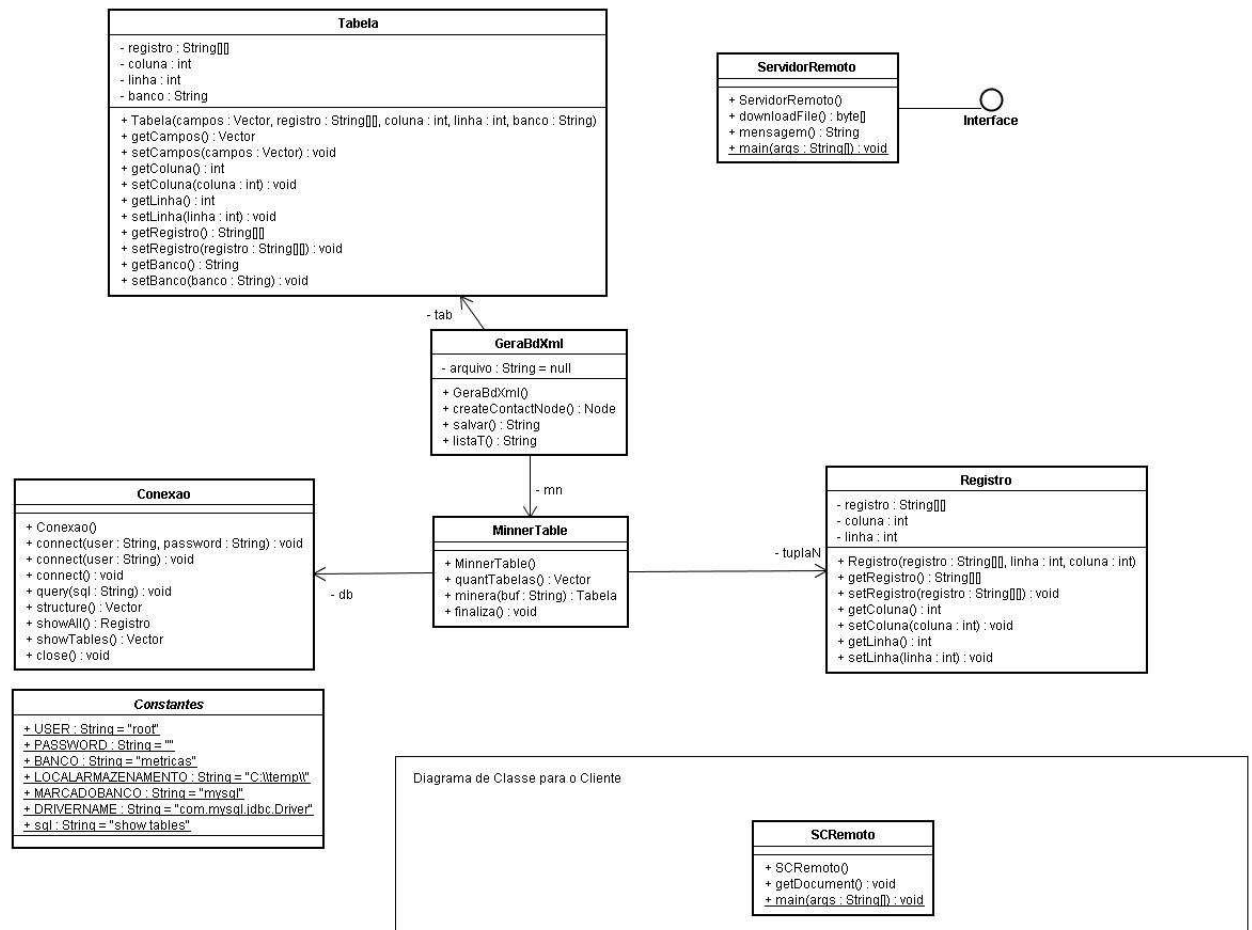
orb.resolve_initial_references("NameService");

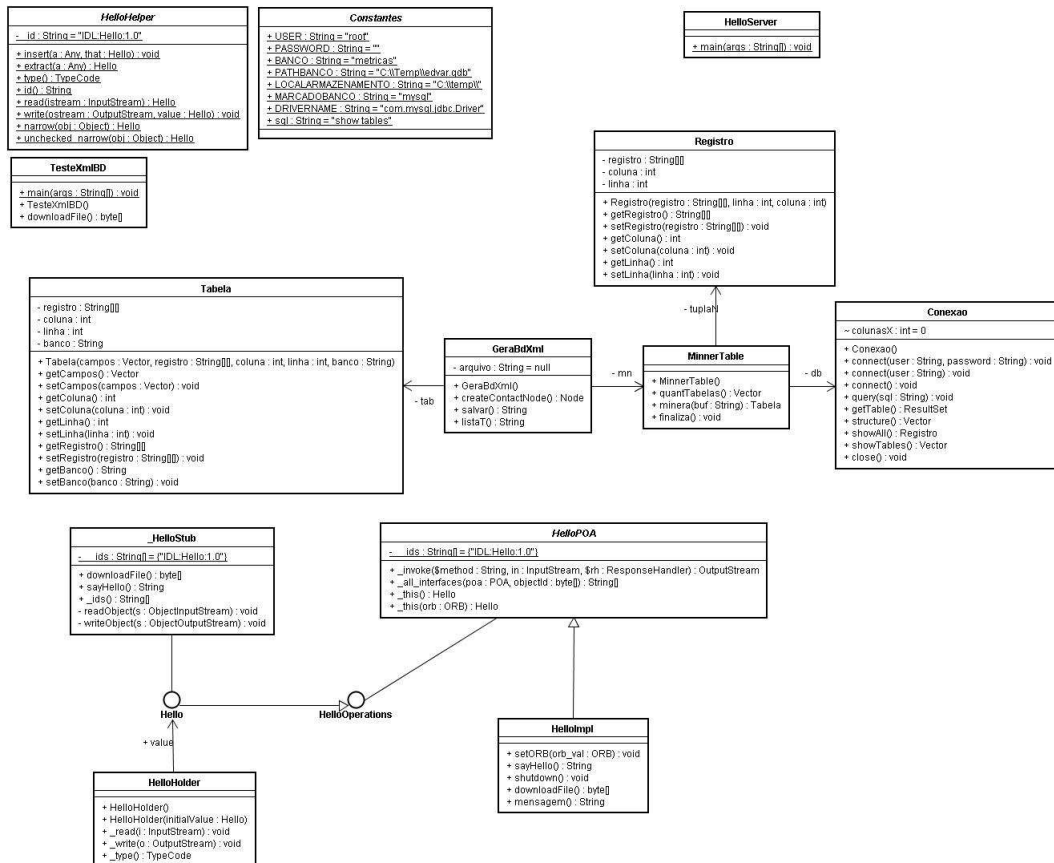
/*NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

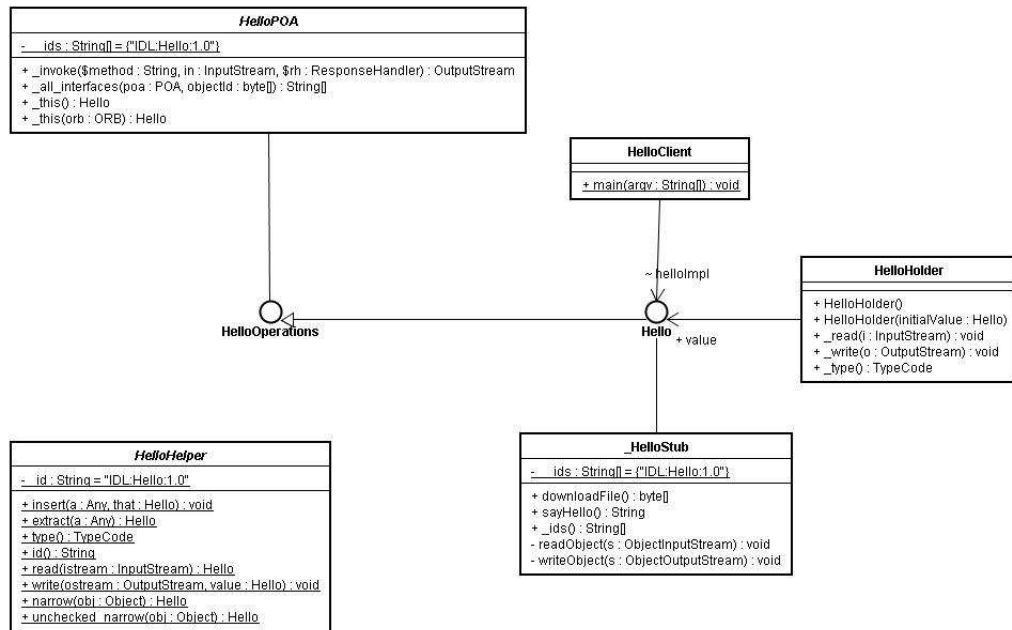
```
NameComponent nc = new NameComponent("FileTransfer"*/  
  
// Use NamingContextExt instead of NamingContext. This is  
  
// part of the Interoperable naming Service.  
  
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);  
  
// resolve the Object Reference in Naming  
  
String name = "Hello";  
  
helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));  
  
System.out.println("Obtained a handle on server object: " + helloImpl);  
  
System.out.println(helloImpl.sayHello());  
  
byte[] filedata = helloImpl.downloadFile();  
  
String nomedoArquivo="Metricas.xml";  
  
File file = new File(nomedoArquivo);//argv[0]);  
  
BufferedOutputStream output = new  
  
BufferedOutputStream(new FileOutputStream(file.getName()));  
  
output.write(filedata,0,filedata.length);  
  
output.flush();  
  
output.close();  
  
//helloImpl.shutdown();  
  
catch (Exception e)  
  
System.out.println("ERROR : " + e) ;  
  
e.printStackTrace(System.out);
```

B Diagramas de Classe

Aqui serão apresentados os diagramas de classe dos programas utilizados neste trabalho. Primeiramente, é mostrado o diagrama das classes utilizadas na implementação baseada no middleware RMI. Em seguida, é mostrados o diagrama de classes da implementação do servidor CORBA, e por fim, o diagrama de classes da implementação do cliente CORBA.

Figura B.1: Diagrama de Classes - *Middleware* RMI - Servidor e Cliente

Figura B.2: Diagrama de Classes - *Middleware* CORBA - Servidor

Figura B.3: Diagrama de Classes - *Middleware* CORBA - Cliente