A Tecnologia *Web Services* e sua Aplicação num Sistema de Gerência de Telecomunicações

Simone da Silva Amorim

Dissertação de Mestrado Profissional

Instituto de Computação
Universidade Estadual de Campinas

A Tecnologia *Web Services* e sua Aplicação num Sistema de Gerência de Telecomunicações

Simone da Silva Amorim

10 de março de 2004

Banca Examinadora:

- **Prof. Dr. Edmundo Roberto Mauro Madeira (Orientador)**
  IC - Unicamp
- **Prof. Dr. Maurício Ferreira Magalhães**
  FEEC - Unicamp
- **Prof. Dr. Geovane Cayres Magalhães**
  IC - Unicamp
- **Profa. Dra. Claudia Maria Bauzer Medeiros (Suplente)**
  IC- Unicamp

A Tecnologia *Web Services* e sua Aplicação num Sistema de Gerência de Telecomunicações

Campinas, 10 de março de 2004.

Prof. Dr. Edmundo Roberto Mauro Madeira
(Orientador)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Computação na área de Engenharia de Computação

Dedico esta monografia à memória do meu pai, à minha mãe, pela força e confiança nas minhas decisões, à minha família pelo incentivo e paciência e a todos os amigos que nunca deixaram de acreditar no meu sucesso.

Agradecimentos

A Deus, em primeiro lugar, pela minha vida, saúde e trabalho, pela oportunidade de realizar este trabalho e pelo amparo nos momentos mais difíceis;

Ao meu pai Álvaro (*in memorian*) pelas lições que me ajudaram a vencer os obstáculos;

À minha mãe Reinalda e ao meu irmão Sidney pelo apoio as minhas decisões e pelo incentivo para enfrentar os desafios;

À minha família em São Vicente – São Paulo que me recebeu de braços abertos e me ajudou a superar os momentos de dificuldades e incertezas;

A todos os meus familiares que souberam compreender a minha ausência e por todo o apoio que me deram, não me deixando desanimar;

Ao meu orientador Prof. Dr. Edmundo R. M. Madeira, principalmente pela paciência, pelas sugestões e ensinamentos, pela confiança e pela seriedade com que conduziu todo o período de orientação;

A todos os amigos e colegas da 1ª turma de Mestrado Profissional da UNICAMP pela convivência, apoio e amizade. Um agradecimento especial a Isana Luzia de Campos;

Aos companheiros do projeto CPqD Gerência de Planta pelos conselhos e sugestões. Em especial a Gerson Mizuta Weiss, Marta Duarte Teixeira e Liliana Kasumi Sasaoka que colaboraram diretamente para a conclusão deste trabalho;

A toda comunidade do Instituto de Computação da UNICAMP;

O desenvolvimento deste trabalho foi parcialmente custeado com recursos do Funttel - Fundo para o Desenvolvimento Tecnológico das Telecomunicações, repassado à Fundação CPqD através da autorização da Portaria no. 581 de 08/10/2001.

Resumo

A tecnologia dos *Web Services* tem se destacado nos últimos tempos no meio computacional como uma revolução em termos de interoperabilidade de sistemas heterogêneos. Esta tecnologia tem sido considerada como a evolução da arquitetura de *middlewares* tradicionais e ganhou o apoio dos grandes produtores de *software* do mercado que a tratam como um novo paradigma no desenvolvimento de sistemas. Eles têm feito grandes esforços em conjunto com os órgãos de padronização de tecnologias no sentido de criar normas padrões, de forma que os *Web Services* possam ser utilizados por todos e que realmente seja obtido um grande nível de interoperabilidade entre as aplicações.

Esta dissertação faz um estudo dos *Web Services* apresentando os conceitos desta tecnologia e fazendo uma comparação com algumas características de outras tecnologias de sistemas distribuídos como CORBA, DCOM e JavaRMI, além de implementar um estudo de caso onde avalia o impacto da aplicação dos *Web Services* na integração de dois sistemas de gerência de telecomunicações.

Abstract

Nowadays the *Web Services* technology has been highlighted as a revolution in terms of interoperability between heterogeneous systems. It has been considered as the evolution of traditional middlewares and gained the support of great software vendors of the market that deal with it as a new paradigm in the development of distributed systems. They have done great efforts together the technology standardization agencies with the goal to create new standards, in order *Web Services* can be used by all and a high level of interoperability between applications is really gotten.

This dissertation presents a study of the *Web Services* describing the concepts of this technology and making a comparison to other middleware technologies as CORBA, DCOM and JavaRMI, besides implementing a case study where it evaluates the impact of the *Web Services* application in the integration of two telecommunication management systems.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ANATEL -Agência Nacional de Telecomunicações

API - Application Programming Interface

BPEL4WS - Business Process Execution Language for Web Services

BPMI - Business Process Management Initiative

BPML - Business Process Management Language

BTP - Business Transaction Protocol

CGI - Common Gateway Interface

CLSID - Class Identifier

COM - Component Object Model

CORBA - Common Object Request Broker Architecture

DCE RPC - Distributed Computing Environment Remote Procedure Call

DCOM - Distributed Component Object Model

DII - Dynamic Invocation Interface

DLL - Dynamic Linked Libraries

DOM - Document Object Model

DNS - Domain Name Service

DSI - Dynamic Skeleton Interface

FTP - File Transfer Protocol

GAT – Gerenciador de Atributos

GIF - Graphic Interchange Format

GIOP - General Inter-ORB Protocol

HTTP - Hypertext Transfer Protocol

HTTP-R - Hypertext Transfer Protocol Reliable

IETF - Internet Engineering Task Force

IIS - Internet Information Server

JAXP - Java API for XML Parsing

JMS - Java Messaging Service

JNDI - Java Naming and Directory Interface

JNI - Java Native Interface

JPEG - Joint Photographers Expert Group

JRMP - Java Remote Method Protocol

JSP - Java Server Pages

JTA - Java Transaction API

JTS - Java Transaction Server

JVM - Java Virtual Machine

LAN - Local Area Network

LDAP - Lightweight Directory Access Protocol

MIDL - Microsoft Interface Definition Language

MSMQ - Microsoft Message Queue

MTS - Microsoft Transaction Server

MWF - Map Window File

NDS - Novell Directory Service

NIS - Network Information Service

NNTP - Network News Transport Protocol

OASIS - Organization for the Advancement of Structured Information Standards

OLE - Object Linking and Embedding

OMG - Object Management Group

ORB - Object Request Broker

ORPC - Object Remote Procedure Call

OSF - Open Software Foundations

OTS - Object Transaction Service

PDA - Personal Digital Assistant

PKI - Public Key Infrastructure

RMI - Remote Method Invocation

RPC - Remote Procedure Call

SAGRE - Sistema Automatizado de Gerência de Rede Externa

SAML - Security Assertion Markup Language

SAX - Simple API for XML

SII - Static Invocation Interface

SMTP - Simple Mail Transfer Protocol

SOAP - Simple Object Access Protocol

SSL - Secure Socket Layer

SSR - Sistema de Supervisão Remota

SVG - Scalable Vector Graphics

TCP/IP - Transmission Control Protocol/Internet Protocol

TPCI - Telefones Públicos de Cartão Indutivo

UDDI - Universal Description, Discover, and Integration

UDP - User Datagram Protocol

UML - Unified Modeling Language

URL - Uniform Resource Locator

VPN - Virtual Private Network

W3C - World Wide Web Consortium

WAN - Wide Area Network

WSCI - Web Services Coreography Interface

WSDD - Web Service Deployment Descriptor

WSDL - Web Service Description Language

WS-I - Web Services Interoperability Organization

XML - Extensible Markup Language

Capítulo 1

Introdução

Nos últimos anos o modelo de arquitetura orientada a serviços vem despertando a atenção dos desenvolvedores de *software* com a promessa de trazer grandes ganhos para a comunicação entre os sistemas de computação existentes hoje. Esta arquitetura pode ser definida como uma arquitetura de *software* que relaciona os componentes de um sistema em um ambiente distribuído onde são disponibilizados serviços que podem ser acessados dinamicamente através de uma rede. Ela apresenta como seus principais benefícios [TYAGI2003]:

- Enfoque nas Regras de Desenvolvimento: Esta arquitetura define a aplicação com múltiplas camadas, onde cada camada tem um conjunto de regras específicas para desenvolvedores. Na camada de serviço o desenvolvimento é focado no formato dos dados, na lógica de negócios, nos mecanismos de persistência, no controle de transações, etc. Na camada do cliente o foco está nas tecnologias de interfaces com os usuários como Swing, Java Server Pages, etc. As empresas podem se beneficiar com o uso de mão de obra especializada para desenvolver cada módulo otimizando o processo de desenvolvimento.

- Facilidade de Testes: Os serviços são publicados em interfaces que são facilmente testadas através de testes de unidade. Podem ser usadas ferramentas para validar o serviço independentemente de qualquer aplicação que use este serviço, podendo refletir em um aumento da qualidade do mesmo.

- Alta Disponibilidade: Por causa da transparência de localização, múltiplos servidores podem executar múltiplas instâncias do serviço e a solicitação do cliente pode ser redirecionada para um servidor que possa atendê-lo melhor sem que ele precise se preocupar com isto.

Uma tecnologia que implementa a maioria das características desta arquitetura é a de *Web Services* que segundo Geraldo Costa [COSTA2002] "propõe a exposição das transações e das regras de negócios por meio de protocolos que podem ser acessados e entendidos por qualquer linguagem de programação, em qualquer sistema operacional, rodando em qualquer dispositivo". Desta forma, os *Web Services* são um caminho para a redução de custos através da redução da redundância dos dados e serviços. Por exemplo, uma empresa com três departamentos diferentes que utilizam o mesmo conjunto de funções distribuídas em diferentes sistemas, podem acessar através dos *Web Services* estes serviços centralizados em um único lugar mesmo que cada sistema pertença a uma plataforma diferente em cada departamento.

Os *Web Services* criam um conjunto de funções empacotadas e distribuídas em algum lugar da rede, para que outros programas possam usá-las livremente. Outras arquiteturas como CORBA (*Common Object Request Broker Architecture*), JavaRMI *(Java Remote Method Invocation)* e o DCOM (*Distributed Component Object Model*) também têm esta proposta, mas terão dificuldades para se tornarem padrão por diversos motivos. Entre eles pode-se dizer que a dificuldade de comunicação com o código legado de outras plataformas, a falta de suporte para linguagens de programação distintas, a complexidade das plataformas e a reescrita de código aumentam consideravelmente os custos de migração de um projeto.

Os *Web Services* trazem ganhos exatamente nas dificuldades das outras tecnologias que citamos anteriormente, já que utilizam o protocolo HTTP (*Hypertext Transfer Protocol*), que é amplamente utilizado hoje pelos sistemas para transportar os dados, além de utilizarem o padrão XML (*Extensible Markup Language*) e o protocolo SOAP (*Simple Object Access Protocol*) para envio e recebimento das mensagens oferecendo suporte para várias linguagens de programação. Desta forma os *Web Services* atuam facilitando o processo de integração entre os sistemas através de sua maneira simples de trabalhar e com isso vêem conquistando grandes espaços nas empresas.

1.1 Motivação

A tecnologia dos *Web Services* está sendo apresentada com o objetivo de realizar o sonho antigo de fazer as aplicações conversarem umas com as outras, seja qual for a plataforma de *hardware* ou de *software*. Com o objetivo de realizar este sonho a padronização desta tecnologia, que se encontra em sua fase inicial, está em discussão em diversos fóruns como o W3C (*World Wide Web Consortium*) [W3C], o IETF (*Internet Engineering Task Force*) [IETF] e o WS-I (*Web Services Interoperability Organization*) [WSI].

Embora os *Web Services* estejam ainda em um estado embrionário, a indústria de tecnologia trata de criar ferramentas para construção de um novo mundo de interoperabilidade. Diante deste contexto ressalta-se a necessidade do estudo desta tecnologia que enfrenta o desafio de integrar diversas aplicações, principalmente abordando os aspectos práticos e o impacto nos sistemas corporativos, assim como a necessidade de examinar e propor as adaptações necessárias para contribuir para o amadurecimento desta tecnologia.

1.2 Objetivo

O objetivo deste trabalho é estudar a tecnologia dos *Web Services* e os seus mecanismos de interconexão, fazendo uma comparação de suas características com as características das principais tecnologias de sistemas distribuídos existentes no mercado. Além de implementar um estudo de caso para a integração de dois sistemas de gerência de rede de telecomunicações no CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) e estudar qual o impacto da aplicação desta tecnologia na interface para disponibilização dos serviços nesta integração.

1.3 Organização do Trabalho

O Capítulo 2 apresenta os principais conceitos da tecnologia dos *Web Services* descrevendo a sua arquitetura e seu funcionamento incluindo os conceitos de SOAP, WSDL (*Web Service Description Language*) e UDDI (*Universal Description, Discover, and*

Integration), colocando as suas características principais, bem como suas vantagens e desvantagens. No final são abordados quais os principais produtos que existem no mercado.

No Capítulo 3 são descritas três das principais tecnologias que se destacam no panorama dos sistemas distribuídos, apresentando de forma geral os conceitos de DCOM, CORBA e JavaRMI. Em seguida traz uma comparação entre as principais características destas tecnologias e como cada uma delas se apresenta na tecnologia dos *Web Services*.

Já no Capítulo 4 é discutido o desenvolvimento de um protótipo de integração entres os sistemas CPqD Gerência de Planta e CPqD Supervisão Remota utilizando a tecnologia dos *Web Services*. É apresentado um projeto onde foi implementada uma simulação que contém desde a sua definição e as tecnologias utilizadas, passando pela descrição dos métodos desenvolvidos em sua interface e como esses métodos trabalham, apresentando ainda algumas conclusões sobre a adequação do funcionamento dos *Web Services* para o contexto de interoperabilidade dos sistemas apresentados.

Por fim, o Capítulo 5 refere-se à conclusão a respeito deste trabalho, além das contribuições e das sugestões para trabalhos futuros.

Capítulo 2

A Tecnologia *Web Services*

Com a evolução das redes de computadores surgiram as aplicações distribuídas. Em princípio o processamento era realizado somente no servidor sendo as aplicações inicializadas por um cliente que estava interligado a ele através da rede. Mais adiante, com o aparecimento da programação orientada a objetos surgiram novos *middlewares* como CORBA, DCOM e RMI, onde o processamento passou a ser distribuído entre vários servidores. Já no momento atual com o avanço da Internet e dos protocolos de comunicação, além de novos padrões como o XML, surgiram os *Web Services* com a proposta de integrar sistemas heterogêneos.

Através de um conjunto de novos conceitos de interoperabilidade como o XML, o SOAP, o WSDL, e o UDDI, os *Web Services* vieram facilitar a comunicação entre as aplicações que residem em múltiplas plataformas, usando diferentes modelos de objetos e baseados em linguagens diferentes. Partindo destas definições, abordamos neste capítulo os principais fundamentos desta tecnologia, explicando o seu funcionamento e, ao final, apresentando alguns dos principais produtos que existem no mercado hoje.

2.1 Definição

Segundo Kreger [KREGER2001] um *Web Service* é uma interface que descreve uma coleção de operações que são acessíveis pela rede através de mensagens XML padronizadas. Ele é um serviço descrito usando um padrão com uma notação formal XML chamada de descrição do serviço. Esta descrição cobre todos os detalhes necessários para interagir com o serviço, incluindo os formatos das mensagens, os protocolos de transporte e as localizações de cada serviço. A interface esconde os detalhes da execução do serviço, permitindo que seja usado independentemente da plataforma de *hardware* ou de *software*

em que esteja implementado e também independentemente da linguagem de programação que foi escrita. Estas características fazem com que as aplicações baseadas em *Web Services* sejam fracamente acopladas e orientadas a serviços, facilitando o uso de vários serviços em conjunto para executar operações complexas.

2.2 Arquitetura dos *Web Services*

A arquitetura dos *Web Services* é baseada na interação de três personagens: Provedor de Serviços, Consumidor de Serviços e Registro dos Serviços. A interação destes personagens envolve as operações de publicação, pesquisa e ligação. Vejamos a definição de cada uma destas funções [KREGER2001]:

Provedor de serviços **à** O provedor de serviços é a entidade que cria o *Web Service*. Ele disponibiliza o serviço para que alguém possa utilizá-lo. Mas para que isto ocorra, ele precisa descrever o *Web Service* em um formato padrão, que seja compreensível para qualquer um que precise usar esse serviço e, também, publicar os detalhes sobre seu *Web Service* em um registro central que esteja disponível.

Consumidor de serviços **à** Qualquer um que utilize um *Web Service* criado por um provedor de serviços é chamado de consumidor de serviços. Este conhece a funcionalidade do *Web Service*, a partir da descrição disponibilizada pelo provedor de serviços, recuperando os seus detalhes através de uma pesquisa sobre o registro publicado. Através desta pesquisa também o consumidor de serviços pode obter o mecanismo para ligação com este *Web Service*.

Registro dos serviços **à** Um registro de serviços é a localização central onde o provedor de serviços pode relacionar seus *Web Services*, e no qual um consumidor de serviços pode pesquisá-los. O registro dos serviços contém informações como detalhes de uma empresa, quais os serviços que ela fornece e a descrição técnica de cada um deles.

Portanto, o provedor de serviços define a descrição do serviço para o *Web Service* e publica esta para o consumidor de serviços no registro de serviços. O consumidor de serviços utiliza a descrição do serviço publicada para se ligar ao provedor de serviços e invocar ou interagir com a implementação do *Web Service*. A Figura 2.1 mostra o relacionamento entre os serviços:

Figura 2.1: Modelo de relacionamento entre os serviços.

2.2.1 Operações na Arquitetura de *Web Services*

Sobre esta arquitetura se executam operações fundamentais para utilização dos *Web Services* que são: Localização, Publicação e Ligação, sendo que cada uma delas precisa ser padronizada para que o serviço possa ser utilizado por qualquer aplicação independente do tipo de linguagem na qual esta foi escrita e a plataforma onde ela está sendo executada.

A Publicação da descrição do serviço disponibiliza informações sobre o serviço utilizando a WSDL, que é uma linguagem padrão que utiliza o formato XML.

Na Pesquisa, o Consumidor de Serviços obtém a descrição do serviço consultando o Registro dos Serviços. Este registro é feito através de um serviço de registros padrão que é o UDDI que armazena os detalhes dos *Web Services* disponibilizados.

Na Ligação, o Consumidor de Serviços invoca a interação com o serviço em tempo de execução usando os detalhes de ligação obtidos na descrição deste serviço. Para fazer esta ligação ao serviço utiliza-se um protocolo padrão de aplicações escrito em XML, o SOAP que é usado para trocar informações entre as aplicações.

2.2.2 A Pilha de Camadas dos *Web Services*

As operações de Publicação, Pesquisa e Ligação podem ser representadas através de uma pilha mostrada na Figura 2.2. As configurações superiores das camadas são construídas com base nas funcionalidades fornecidas pelas camadas mais baixas. O texto na esquerda representa as tecnologias padrão que se aplicam nessa camada da pilha [KREGER2001].

Figura 2.2: Pilha de camadas dos *Web Services*

As camadas inferiores desta pilha são mais maduras e padronizadas do que as camadas mais altas da pilha. A forma como os *Web Services* serão utilizados definirão a orientação para o desenvolvimento e a padronização dos níveis mais elevados.

Rede de Transporte

Essa camada da pilha de *Web Services* é responsável pela disponibilização dos serviços por intermédio de algum dos protocolos de transporte disponíveis na rede, como HTTP, SMTP (*Simple Mail Transfer Protocol*), FTP (*File Transfer Protocol*) e outros. Hoje, o HTTP é o protocolo de comunicação mais amplamente utilizado, por isso ele é recomendado como principal protocolo de rede para os *Web Services* na Internet. Entretanto, para os *Web*

Services que podem ser acessados dentro de uma Intranet, pode-se utilizar tecnologias de rede alternativas que podem ser escolhidas baseadas em outros requisitos, incluindo a segurança, a disponibilidade, o desempenho e a confiabilidade.

Mensagens XML

A próxima camada da pilha de *Web Services* é a troca de mensagens XML. Esta camada define o formato de mensagem usado na comunicação entre aplicações. O padrão usado com regularidade pelos *Web Services* é o SOAP, um protocolo com base em XML para a troca de informações entre aplicações, independentemente do sistema operacional, do ambiente operacional e do modelo do objeto.

Descrição dos Serviços

A camada descrição do serviço fornece um mecanismo ao provedor de serviços a fim de descrever as funcionalidades proporcionadas pelos *Web Services*. Nela utiliza-se a linguagem WSDL para detalhar todas as características de cada serviço. Basicamente, o documento WSDL para um *Web Service* define os métodos que estão presentes no serviço, os parâmetros de entrada/saída para cada um dos métodos, os tipos de dados, o protocolo de transporte usado e a URL(*Uniform Resource Locator*) da máquina onde o *Web Service* será hospedado.

Publicação e Pesquisa dos Serviços

Em vez de publicar o documento WSDL para cada possível cliente, um provedor de serviços será mais eficiente se publicar as informações disponíveis sobre seu *Web Service* em um registro central que esteja disponível publicamente para os consumidores de serviços interessados. Além de publicar apenas a descrição de seu *Web Service*, ele pode também publicar informações relacionadas ao negócio. Da mesma forma, os consumidores de serviços podem querer encontrar em um único lugar, diferentes *Web Services* que sejam disponibilizados pelos provedores do serviço para avaliá-los antes de integrar estes serviços

às suas aplicações. Por isso, utiliza-se um registro central, que é o UDDI, onde os provedores e os consumidores dos serviços trabalham em conjunto para publicar e recuperar as informações desejadas.

2.3 Detalhamento das Tecnologias

2.3.1 SOAP

O SOAP é um protocolo baseado em XML que permite que dois programas se comuniquem. Ele está no topo da camada de transporte de dados e não está vinculado a nenhum protocolo de transporte em particular. O SOAP serve basicamente para a troca de informações e/ou dados através de objetos criados em diversas linguagens de programação como *Visual Basic*, Java, C++, etc. Por exemplo, a camada SOAP pode ser adicionada a um objeto Java e CORBA criando um ambiente de computação distribuído comum para os dois objetos, tornando os objetos CORBA acessíveis pelos objetos Java e vice-versa, sem envolver as particularidades de cada protocolo [W3S].

O SOAP contém menos recursos que os outros protocolos de computação distribuídos, mas existe uma grande mobilização dos fornecedores de tecnologia para desenvolvê-lo e transformá-lo em um padrão robusto que seja aceito pela maioria dos produtos existentes no mercado.

O servidor SOAP tem a habilidade de construir e analisar a mensagem SOAP e transmiti-la através da rede. Ele funciona como uma biblioteca em tempo de execução em uma linguagem de programação específica que pode ser usada para encapsular estas funções dentro de uma API (*Application Programming Interface*).

A aplicação cliente cria uma mensagem SOAP invocando a operação do *Web Service* interagindo com um protocolo de transporte de dados subjacente (por exemplo o HTTP) para emitir a mensagem sobre a rede. Ao chegar no servidor SOAP, a mensagem é convertida em objetos de uma linguagem de programação específica na qual está escrito o

Web Service, usando esquemas de codificação que são encontrados dentro da própria mensagem. Este *Web Service* executa o serviço e formula a resposta enviando esta para o servidor SOAP que transforma esta resposta em uma mensagem SOAP tendo o cliente do serviço como destinatário. Ele envia a resposta através da rede convertendo o conteúdo da mensagem de XML em objetos da linguagem de programação determinada. A mensagem de resposta é então apresentada à aplicação [KREGER2001].

O SOAP proporciona várias vantagens, em relação a outros sistemas de comunicação, entre eles:

- O SOAP pode atravessar *firewalls* com facilidade.
- Os dados do SOAP são estruturados usando XML. Portanto, as mensagens podem ser compreendidas por quase todas as plataformas de *hardware*, sistemas operacionais e linguagens de programação.
- O SOAP pode ser usado, potencialmente, em combinação com vários protocolos de transporte de dados, como HTTP, SMTP e FTP.
- O SOAP mapeia satisfatoriamente para o padrão de solicitação/resposta HTTP.
- Pode ser usado tanto de forma anônima como com autenticação (nome/senha).

As principais desvantagens do SOAP são:

- Falta de interoperabilidade entre ferramentas de desenvolvimento do SOAP. Embora o SOAP tenha amplo suporte, ainda existem problemas de incompatibilidades entre diferentes implementações do SOAP.
- Mecanismos de Segurança Imaturos. O SOAP não define mecanismo para criptografia do conteúdo de uma mensagem SOAP, o que evitaria que outros tivessem acesso ao conteúdo da mensagem.
- Não existe garantia quanto à entrega da mensagem. Quando uma mensagem estiver sendo transferida, se o sistema falhar, ele não saberá como reenviar a mensagem.
- Um cliente SOAP não pode enviar uma solicitação a vários servidores, sem enviar a solicitação a todos os servidores.

As aplicações permitem que o SOAP possa transpor os *firewalls* com facilidade, pois ele pode ser usado com o HTTP. Isto permite que os *softwares* que aceitam o SOAP estejam

disponíveis internamente e externamente na rede. Esta característica pode ser vista como vantagem e também como desvantagem, já que pode causar um sério problema de segurança, onde as aplicações do SOAP seriam acessíveis por partes não autorizadas [HENDRICKS2002].

De acordo com o W3Schools [W3S] a estrutura da mensagem SOAP é definida em um documento XML que contém os seguintes elementos:

- *Envelope*: Identifica o documento XML como uma mensagem SOAP e é responsável por definir o conteúdo da mensagem;
- *Header* (opcional): Contém os dados do cabeçalho;
- *Body*: Contém as informações de chamada e de resposta ao servidor;
- *Fault*: Contém as informações dos erros ocorridos no envio da mensagem. Esse elemento só aparece nas mensagens de resposta do servidor.

A estrutura da mensagem SOAP possui a seguinte forma:

```
<SOAP-ENV:envelope>
<!--Esse é o elemento raiz do SOAP e define que essa é uma mensagem SOAP-
->
<SOAP-ENV:header>
<!--Especifica informações específicas como autenticação, etc.(é
opcional)-->
</SOAP-ENV:header>
<SOAP-ENV:body>
<!--O elemento BODY contém o corpo da mensagem-->
<SOAP-ENV:fault>
<!--O elemento FAULT contém os erros que podem ocorrer-->
</SOAP-ENV:fault>
</SOAP-ENV:body>
</SOAP-ENV:envelope>
```

A seguir temos um exemplo de uma mensagem SOAP utilizada na implementação que foi definida no capítulo 4:

1. O elemento *envelope* sendo a raiz que define a mensagem SOAP:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
</soapenv:Envelope>
```

2. O elemento *body* contendo o corpo da mensagem, especificando a chamada para a função getRendaMes e os seus parâmetros cnl, mcdu, prefixo, mes e ano:

```
 <soapenv:Body>
  <getRendaMes
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <cnl xsi:type="xsd:string">41031</cnl>
   <mcdu xsi:type="xsd:string">4020</mcdu>
   <prefixo xsi:type="xsd:string">423</prefixo>
   <mes xsi:type="xsd:string">12</mes>
   <ano xsi:type="xsd:string">2003</ano>
  </getRendaMes>
 </soapenv:Body>
```

2.3.2 WSDL

Um documento WSDL descreve os serviços do *Web Service* através do XML, fornecendo uma documentação do serviço para que possíveis clientes possam utilizá-lo de forma automatizada. A WSDL apresenta a definição de um *Web Service* em duas partes. A primeira representa uma definição abstrata independentemente do protocolo de transporte de alto nível, enquanto a segunda representa uma descrição da ligação específica para o transporte na rede [KREGER2001].

Figura 2.3: Elementos do WSDL.

A Figura 2.3 demonstra que um WSDL contém os seguintes elementos:

- Tipo de Porta (*portType*): Os elementos *portType* contêm um conjunto de operações representadas como elementos *operation* que possui um atributo *name* e um outro atributo opcional que especifica a ordem dos parâmetros usados nas operações que estão presentes em um *Web Service*. Além disso, ele pode ter apenas uma mensagem de entrada e uma mensagem de saída, da mesma maneira que uma chamada de método normal.
- Mensagem (m*essage*) **à** Um elemento *message* contém uma definição dos dados a serem transmitidos. É semelhante ao parâmetro na chamada do método.
- Tipo (t*ype*) **à** O elemento *type* contém os tipos de dados que estão presentes na mensagem.
- Ligação (b*inding*) **à** O elemento *binding* mapeia os elementos *operation* em um elemento *portType*, para um protocolo específico.
- Serviço (*service*) e Porta (*port*) **è** Os elementos *service* e *port* (contido no *service*) incluem a localização da implementação do serviço na rede, ou seja, contém a informação para onde enviar a solicitação do serviço.

Para que uma operação possa ser executada em uma rede, devemos usar um protocolo de transporte específico para enviar os dados pela rede. É onde o elemento *binding* atua definindo um protocolo particular (SOAP, HTTP GET, HTTP POST, e outros) e ligando-o aos elementos *portType*, *message* e *types* mencionados acima. Podemos, portanto ligar a definição abstrata a vários protocolos de transporte, o que significa que uma definição de WSDL pode permitir que definamos um *Web Service*, independentemente de transporte.

De acordo com a estrutura definida em [GUNZER2002] o elemento raiz de qualquer documento WSDL é o elemento <definitions>. Os elementos contêm referências um para o outro como mostra a Figura 2.4 a seguir. A descrição ao lado da seta representa o nome do atributo dos elementos que contém a referência.

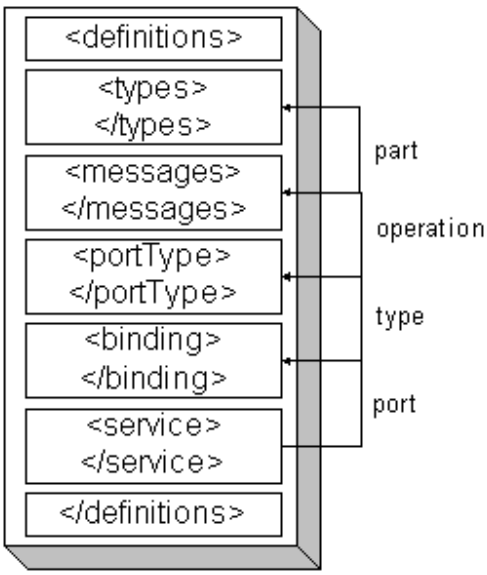


Figura 2.4: Estrutura do WSDL.

A seguir temos alguns exemplos de trechos de código WSDL que fazem parte da implementação feita no capítulo 4 e que demonstram as definições vistas anteriormente. Começamos com o elemento *portType* definindo a operação "getRendaMes" que tem a mensagem "getRendaMesRequest" como entrada e que produz a mensagem "getRendaMesResponse" como saída. Deste modo, definimos a operação *request/response*:

```
<wsdl:portType name="Consultas">
<wsdl:operation name="getRendaMes" parameterOrder="cnl mcdu prefixo mes
ano">
<wsdl:input message="impl:getRendaMesRequest" name="getRendaMesRequest"/>
<wsdl:output message="impl:getRendaMesResponse"
name="getRendaMesResponse"/>
</wsdl:operation>
</wsdl:portType>
```

Continuamos com o elemento *message* onde estão especificados os dados que serão transmitidos entre o cliente e o serviço. O elemento <part> especifica o conteúdo da mensagem representando os parâmetros que são passados para o serviço e a resposta que ele retorna.

```
<wsdl:message name="getRendaMesRequest">
<wsdl:part name="cnl" type="xsd:string" />
<wsdl:part name="mcdu" type="xsd:string" />
<wsdl:part name="prefixo" type="xsd:string" />
```

```
<wsdl:part name="mes" type="xsd:string" />
<wsdl:part name="ano" type="xsd:string" />
</wsdl:message>
<wsdl:message name="getRendaMesResponse">
<wsdl:part name="getRendaMesReturn" type="xsd:string" />
</wsdl:message>
```

O elemento *type* contém as definições de estruturas de dados mais complexas como vetores, etc. Como, em nosso caso, utilizamos apenas tipos de dados primitivos (String), não houve geração do elemento *type* no documento WSDL.

Temos o elemento *binding* que associa o elemento *portType* ao protocolo SOAP através de um elemento de extensão SOAP chamado <wsdlsoap:binding>. Ele fornece dois parâmetros: o protocolo de transporte e o estilo da requisição, que para o nosso caso é "rpc".

```
<wsdl:binding name="ConsultasSoapBinding" type="impl:Consultas">
<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />
```

Por último temos os elementos *service* e *port* que definem a localização real do serviço, sendo que o serviço pode conter várias portas e cada uma delas é específica para o tipo de ligação que foi descrito no elemento *binding*.

```
<wsdl:service name="ConsultasService">
<wsdl:port binding="impl:ConsultasSoapBinding" name="Consultas">
<wsdlsoap:address location="http://cpqd050203:8080/axis/Consultas.jws"/>
</wsdl:port>
</wsdl:service>
```

2.3.3 UDDI

Em seu artigo sobre introdução aos *Web Services*, Gunzer [GUNZER2002] define a UDDI como um padrão projetado para fornecer um diretório de busca para os negócios e seus serviços. Ele atua como mediador do serviço, permitindo que os clientes requisitantes encontrem um fornecedor do serviço apropriado. A sua implementação é semelhante a uma lista telefônica formada pelas seguintes partes:

- Páginas amarelas: Contêm informações organizadas por categoria específica do produto, ou por regiões geográficas.
- Páginas Brancas: Contêm informações sobre os fornecedores de serviços, incluindo o endereço, o contato e os identificadores conhecidos.
- Páginas Verdes: Contêm informações técnicas sobre os serviços dos *Web Services* que são expostos pelo negócio. Por exemplo, explica como fazer a comunicação com eles.

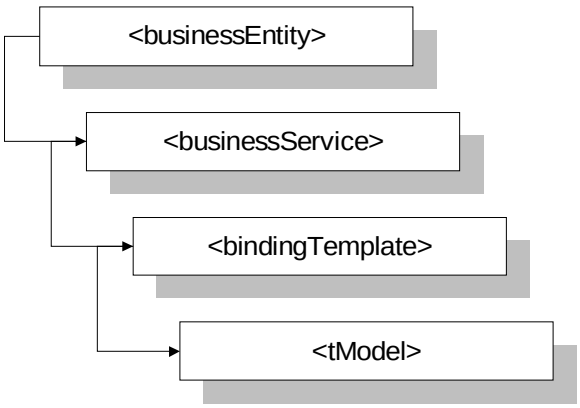


Figura 2.5: Estrutura do UDDI.

O UDDI é constituído por quatro especificações principais (Figura 2.5):

1. *Business Entities*: Contém informações sobre a empresa que publica o *Web Service*. Esta informação contém o nome da empresa, e também pode incluir as informações das URLs dos serviços, os serviços oferecidos, além de informações de contato com a empresa.
2. *Business Services*: Contém informações sobre a descrição de cada serviço oferecido em termos de negócios.
3. *Binding Template*: Contém informações que descrevem como obter o acesso a um serviço, informando quais os pontos de acesso ao serviço chamado através de URLs.
4. *tModels*: Contém informações que descrevem uma especificação técnica do serviço. Por exemplo, os protocolos de rede ou as regras de seqüência.

Através desta estrutura a UDDI consegue definir a maneira de publicar e descobrir as informações sobre os *Web Services*, descrevendo as funcionalidades específicas do serviço disponibilizado por uma empresa, através de uma conexão com a Internet para que outros *softwares* possam utilizá-lo.

2.4 Aplicação dos *Web Services*

Um dos principais objetivos dos *Web Services* é prover a interoperabilidade entre diferentes sistemas, independente da plataforma e da linguagem de programação utilizada por eles, tornando mais fácil a interligação destas aplicações [ARSANJANI2003]. Esta integração pode melhorar os processos de negócios tornando-os automatizados e interligando-os a outros processos que trabalham de forma isolada. Desta forma, uma estratégia bem sucedida de integração pode permitir a diminuição de custos, aumentar a produtividade e trazer novas oportunidades de rendimento. Entretanto, conseguir este sucesso requer um planejamento cuidadoso para evitar que esta integração possa trazer prejuízos ao invés de trazer benefícios.

Permitir que componentes lógicos desenvolvidos de formas diversas trabalhem juntos apresentam alguns problemas naturais, e para que isto seja feito existe a necessidade de vencer estes obstáculos que surgem quando estes sistemas começam interagir. De acordo com o Gartner Group, 80% dos sistemas legados foram escritos na linguagem de programação COBOL, possuem pouca flexibilidade e não foram projetados de forma modular [ARSANJANI2003]. Fazer com que estes sistemas antigos possam interagir com outros, envolve mais do que um simples empacotamento de funções, podendo haver necessidade de adaptá-los para que eles possam ser integrados com outros sistemas baseados em outras plataformas. A arquitetura destes sistemas legados deve ser inventariada para identificar as áreas funcionais e as partes do código que devem ser reestruturadas, e também deve ser feita uma análise de quais informações podem ser expostas para os outros sistemas.

Outro ponto a ser observado é que o modelo dos *Web Services* também pode promover o reuso dos componentes existentes em muitos sistemas integrados, onde cada componente pode representar um serviço distinto, sem sua lógica estar amarrada a um único projeto de integração, e seus componentes podem participar de múltiplos sistemas sem serem modificados, já que melhorar o reuso significa menos esforço, mais benefícios imediatos e aumento da agilidade no negócio.

Portanto, durante todo o ciclo de vida de um projeto de integração, os *Web Services* devem ser cuidadosamente estruturados para que reduzam o risco de exposição indevida de informações dos sistemas e reutilize o máximo possível o código existente, reduzindo a redundância do código e conseqüentemente os custos do projeto. Além disso, deve-se ter bom senso ao projetar estes *Web Services*, já que a estratégia que for adotada para o projeto terá um impacto significativo sobre o negócio.

2.5 Principais Produtos do Mercado

2.5.1 AXIS

A Apache [AXIS] define o AXIS (***A**pache **EX**tensible **I**nteraction **S**ystem*) como um mecanismo de comunicação SOAP. Ele pode ser usado como uma biblioteca do cliente para invocar os serviços SOAP disponíveis em outra máquina ou como uma ferramenta no lado do servidor para executar serviços acessíveis pelo SOAP. No lado do cliente ele fornece uma API para invocar serviços do SOAP RPC emitindo e recebendo mensagens através do protocolo SOAP. No lado do servidor ele fornece um mecanismo para escrever e acessar serviços RPC ou serviços de mensagens, que devem ser hospedados por um *servlet container* (tal como *Apache Tomcat*).

O AXIS é a versão 3.0 do *Apache* SOAP que começou na IBM com o "SOAP4J" e tem o objetivo de ser um sistema mais flexível e capaz de assegurar o tratamento do SOAP definido na especificação do protocolo de XML do W3C. A versão atual do AXIS é escrita em Java, mas uma implementação em C++ no lado do cliente do AXIS está sendo

desenvolvida. Além de funções de suporte para o SOAP, ele inclui também suporte extensivo para WSDL, possui ferramentas que geram classes Java de um WSDL e uma ferramenta para monitorar pacotes TCP/IP.

O AXIS permite a separação entre os problemas de infra-estrutura e o processamento do negócio devido ao fato da execução das tarefas de autenticação e autorização serem executadas em seqüência antes de chamar os serviços de negócio. Ele propicia o processamento específico de uma mensagem SOAP em uma seqüência de operações através da aplicação de um modelo modular que aceita as evoluções das especificações sem causar grandes impactos na arquitetura da plataforma.

O *Apache* AXIS em conjunto com o *Apache Tomcat* e o *Java Development Kit* (JDK) fornecem um conjunto suficiente de utilitários destinados a realizar todas as etapas de desenvolvimento de um projeto sem precisar de bibliotecas adicionais.

2.5.2 .NET

Segundo a Microsoft, .NET é uma plataforma de *software* que conecta sistemas e dispositivos através de várias tecnologias, permitindo o acesso às informações e possibilitando que o usuário interaja com estes dispositivos inteligentes através da Web. Ela é basicamente um conjunto de XML *Web Services* que possibilita que sistemas e aplicativos, novos ou já existentes, conectem seus dados e transações independente do sistema operacional, do tipo de computador, do dispositivo móvel ou de qual linguagem de programação tenha sido utilizada na sua criação [NET].

A plataforma .NET é formada pelos seguintes componentes:

- .NET *Framework e Visual Studio* .NET: Essas são as ferramentas do desenvolvedor usadas para construir aplicações e XML *Web Services*. A .NET *Framework* é o conjunto de interfaces de programação que constitui o núcleo da plataforma .NET. Ela é uma biblioteca de classes que reúne todas as funções normalmente associadas

ao sistema operacional, resolvendo muitos problemas da API do Windows. O *Visual Studio* .NET é um conjunto de ferramentas de desenvolvimento, onde o programador pode criar soluções com a combinação de VB.NET, ASP.NET, C# e Managed C++ (extensão da Microsoft para o C++ na plataforma .NET).

- .NET Server Infrastructure: São servidores de serviços de infra-estrutura que utilizam os XML *Web Services* e se baseiam no sistema operacional do Windows e dos servidores do *Windows Server System*. Entre esses servidores estão: o *Application Center 2000*; o *BizTalk TM Server 2000*; o *Microsoft Host Integration Server 2000*; o *Microsoft Mobile Information 2001 Server* e o *SQL Server*.

- .NET *Building Blocks Services*: Esses serviços constituem um conjunto integrado de XML *Web Services* para controle dos dados. Entre eles está o *Passport* (para identificação do usuário) e serviços de entrega de mensagens, arquivamento, gerenciamento das preferências do usuário, agendamento e outras funções.

- .NET *Device Software*: Esse *software* possibilita que um conjunto de dispositivos tais como Pocket PC, smart-phones, Tablet PC, Desktop PC, entre outros, suporte a plataforma .NET. Ele é constituído do Windows XP, Windows Me, Windows CE, Windows *Embedded*, .NET *Framework* e .NET *Compact Framework*.

2.5.3 Sun ONE

De acordo com a Sun, Sun ONE (Sun Open Net Environment) é uma plataforma baseada em padrões abertos que fornece um conjunto de práticas para construir e disponibilizar *Web Services*. Ela tem como recursos chaves as tecnologias XML e Java para prover interoperabilidade entre as aplicações. A XML fornece estruturas de dados padrão e neutras com relação à plataforma para representar os dados contextuais, e o Java fornece um conjunto de interfaces neutras em relação à plataforma para acessar e usar essas informações [SUN].

A sua arquitetura baseia-se em padrões abertos como UDDI, SOAP, XML e Java. A sua plataforma é composta pelo ambiente operacional Solaris; do *software iPlanet* formado por servidores de aplicações, localização, web, comércio e comunicações; e do ambiente de desenvolvimento integrado Forte. Ela oferece uma plataforma para Internet integrável, escalável e de custo compatível que foi desenhada para integração simples com *hardwares* e *softwares* existentes, independentemente de plataforma ou fabricante.

2.5.4 IBM *WebSphere*

Como definido em [IBM], o *Web Sphere* é um *software* de infra-estrutura que permite o desenvolvimento e a integração de aplicações. Ele proporciona acesso a informação personalizada a vários tipos de usuários e através de diversos dispositivos móveis, além de integrar aplicativos automatizando os processos de negócios para obter maior eficiência operacional e flexibilidade.

Sua plataforma fornece um conjunto de ferramentas de desenvolvimento que criam, implementam e gerenciam o *e-business*, disponibilizando os processos de negócios e transações através de uma infra-estrutura confiável e escalável. Dentre estas ferramentas existe um ambiente operacional central que implementa *Web Services* e permite transações de forma segura. Este ambiente faz parte da família *WebSphere Application Server* que fornece mecanismos de transações altamente escaláveis e de bom desempenho para aplicativos dinâmicos de *e-business*. Nestas ferramentas também existe a família *WebSphere Studio* que permite o desenvolvimento de aplicativos de forma integrada. Este é um ambiente de desenvolvimento projetado para atender as necessidades de desenvolvimento de interfaces Web e aplicativos de servidor, além de desenvolvimento de componentes Java para a integração dos aplicativos. O *WebSphere Studio* permite criar aplicativos em Java e torná-los acessíveis na Web utilizando os padrões dos *Web Services*, e integrando ferramentas visuais para criar adaptadores de *softwares* e desenvolver aplicações para celulares, PDAs (*Personal Digital Assistant*) e computadores portáteis. A plataforma *WebSphere* fornece também outras ferramentas para criação de portais de negócios, serviços de transações e integração de aplicações.

2.6 Considerações Finais

Os *Web Services* estão em destaque no mundo da computação distribuída como uma tecnologia que resolverá os problemas de interoperabilidade dos sistemas, pois utilizam padrões neutros de plataforma, como HTTP e XML, que escondem dos clientes os detalhes de implementação do serviço. No entanto, ainda existe pouca pesquisa no meio acadêmico relacionada a esta tecnologia e os órgãos de padronização ainda estão trabalhando para resolver as falhas existentes e desenvolver um padrão definitivo. Com base nesta situação fizemos nos capítulos seguintes um estudo desta nova tecnologia visando o aspecto da interoperabilidade dos sistemas, avaliando as suas características em relação às outras tecnologias e a aplicação desta ao contexto da integração de dois sistemas de gerência de rede de telecomunicações.

Capítulo 3

Comparação das Tecnologias

Os *Web Services* tem como proposta simplificar o desenvolvimento e a integração de sistemas. Eles são constituídos de uma série de tecnologias e padrões que são combinados para criar um serviço de comunicação entre as aplicações. O serviço resultante desta comunicação pode permitir uma adaptação mais rápida entre aplicações em relação às tecnologias de integração mais tradicionais. Entretanto, é necessário observar quais são as suas características reais e as suas limitações para que estes sejam utilizados de uma forma correta, considerando exatamente onde e como eles podem substituir as arquiteturas dos sistemas distribuídos existentes hoje.

Neste capítulo é apresentada uma breve definição destas arquiteturas tradicionais, e em seguida, é exposta uma análise sucinta das características mais importantes, incluindo a sua expressão na arquitetura dos *Web Services*, dando ênfase aos fatores que podem influenciar a escolha por uma destas plataformas.

3.1 Principais Plataformas de Comunicação

Apresentamos a seguir três das mais conhecidas plataformas de comunicação para sistemas distribuídos. A CORBA que foi desenvolvida pela OMG (*Object Management Group*) [OMG], o DCOM que é uma versão distribuída do modelo COM da Microsoft e a JavaRMI da Sun que é a versão do Java para programação em ambientes distribuídos.

3.1.1 CORBA

O esforço de padronização de múltiplas empresas reunidas no consórcio OMG deu origem à arquitetura CORBA. Esta foi a resposta à necessidade para interoperabilidade entre os

inúmeros produtos e ferramentas de *software* disponíveis para comunicação entre objetos. A CORBA procura criar um ambiente genérico de desenvolvimento de aplicações baseadas em objetos distribuídos, permitindo que as aplicações se comuniquem uma com a outra não importando onde são encontradas e quem as projetou.

A base desta arquitetura é um barramento de objetos, denominado ORB (*Object Request Broker*), que é o *middleware* que estabelece os relacionamentos do servidor com o cliente, sendo através dele que os componentes de *software* se ligam para interoperarem. O ORB intercepta o atendimento e é responsável para encontrar um objeto que possa executar o pedido, para lhe passar os parâmetros, invocar seu método e retorna seus resultados sem que o cliente tenha conhecimento de onde está localizado este objeto, qual a sua linguagem de programação ou o seu sistema operacional. Desta forma, a estrutura de comunicações do ORB permite aos objetos conversarem, não importando quais as técnicas utilizadas para a sua implementação e quais os aspectos específicos da plataforma.

De acordo com Roque [ROQUE2000] as características mais importantes da arquitetura CORBA são a dinâmica dos objetos, a invocação estática, o repositório de interfaces disponíveis, a segurança e a independência das linguagens de alto nível. A Figura 3.1 mostra a arquitetura CORBA.

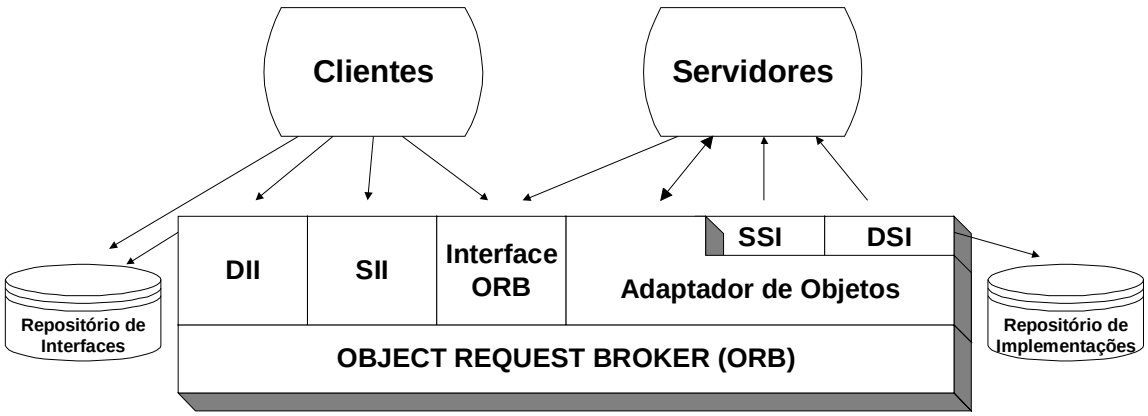


Figura 3.1: Arquitetura CORBA.

Todos os objetos e serviços CORBA são definidos por uma interface escrita na Linguagem de Definição para Interface (IDL). Para publicar um objeto CORBA, os programadores escrevem o código do objeto usando uma linguagem específica e usam a IDL para expressar seus atributos, as operações que o objeto pode realizar, os seus parâmetros de entrada e saída, os eventos que o objeto pode emitir e as exceções.

O Repositório de Interfaces é um componente utilizado pela IDL para armazenar todas as definições dos seus objetos. Ele é o meio pelo qual o ORB permite o acesso distribuído às implementações destes objetos, disponibilizando os elementos da arquitetura de gerenciamento de objetos para as interfaces públicas especificadas na IDL. Esse componente deve estar sempre disponível para o cliente, para o ORB e para as implementações, permitindo a adição, exclusão ou execução de depuração destas interfaces, além de poder invocar suas operações como qualquer outro objeto.

O Adaptador de Objetos é a entidade responsável pela disponibilização do ambiente de suporte do servidor que permite a instanciação e invocação dos objetos. Sua principal função é auxiliar o ORB no despacho de requisições para os objetos e na ativação de um determinado objeto. Ele é considerado o núcleo da comunicação do ORB, sendo um componente chave para a portabilidade da implementação dos objetos. Ele oferece os seguintes serviços: registro de implementação de objetos, criação de instâncias de novos objetos em tempo de execução, geração e gerenciamento de referências dos objetos, capacidade de invocações, ativação e desativação de implementações de objetos, comunicação ao ORB sobre quais os servidores que estão ativos e segurança das interações garantindo a entrega das requisições ao método apropriado.

A Interface de Invocação Estática (*Static Invocation Interface* - SII) é utilizada para acessar uma operação sobre um objeto, onde o cliente precisa chamar o ***stub*** correspondente. Esta invocação é compilada pelo compilador IDL e o cliente conhece quais são as interfaces definidas, os objetos e os métodos disponíveis. Os ***stubs*** são invocados como procedimentos normais de uma linguagem de programação e estão disponíveis na forma de

rotinas de biblioteca, sendo gerados em tempo de compilação da descrição da interface do objeto.

A Interface de Invocação Dinâmica (*Dynamic Invocation Interface* - DII) é utilizada para especificar e construir um pedido em tempo de execução, ou seja, ela permite a descoberta e a construção de uma invocação ao objeto dinamicamente, como também possibilita a inclusão do mesmo sem conhecimento prévio dos métodos disponíveis. O cliente fica responsável por especificar todos os parâmetros e os resultados esperados para que a requisição chegue de forma transparente para o objeto. As desvantagens deste método são o tempo e o tráfego adicionais na rede provocado pelos acessos adicionais ao Repositório de Interfaces para verificarem os parâmetros em tempo de execução.

A Interface de Esqueleto Estática (*Static Skeleton Interface* - SSI) é utilizada no lado do servidor para desempacotar as solicitações e transmiti-las as implementações dos objetos e quando ela recebe os resultados faz o empacotamento para que sejam enviados para o cliente.

A Interface de Esqueleto Dinâmico (*Dynamic Skeleton Interface* - DSI) é utilizada para permitir o tratamento dinâmico de invocação de interfaces, oferecendo seus recursos para os objetos servidores. Neste caso, a implementação de objetos é obtida através de uma interface que provê acesso aos nomes das operações e parâmetros de forma semelhante a interface de invocação dinâmica do lado do cliente. Ela consulta os valores dos parâmetros que vêem na mensagem para obter o objeto e o método destino da invocação.

O desenvolvimento das especificações de serviços de alcance geral foi um dos principais aspectos ressaltados pelo OMG no processo de especificação da CORBA. Estes serviços permitem a criação dos componentes, identificando-os e colocando-os disponíveis no sistema, com o objetivo de complementar a funcionalidade do ORB. A implementação de todos os serviços definidos pelo OMG caminha de forma lenta e apenas uma parte dos objetivos iniciais foram transformados em produtos comerciais. Como plataformas

comerciais se destacam, o Orbix da IONA Technologies e o Visibroker da Inprise [ROQUE2000].

3.1.2 DCOM

Em 1993 para resolver as deficiências da tecnologia OLE (*Object Linking and Embeddign*) a Microsoft lançou uma nova tecnologia de encapsulamento de objetos designada COM (*Component Object Model*). O COM passa então a ser divulgado pela Microsoft como um modelo de programação orientado por objetos com suporte para integração de várias aplicações diferentes e desenhado para facilitar a interoperabilidade do *software*.

Um objeto no modelo COM é uma entidade funcional que obedece ao princípio de encapsulamento de orientação a objeto. Os clientes não manipulam os objetos diretamente. Ao invés disso, o objeto exporta para os seus clientes vários conjuntos de ponteiros de funções, conhecidos como interfaces. Uma interface é um ponteiro para uma tabela de ponteiros de funções. Um objeto pode suportar várias interfaces. Todos os objetos COM devem suportar a interface mais básica, *IUnknown* (por convenção, os nomes de interfaces iniciam com "I"), que suporta três métodos que fornecem a funcionalidade básica para todos os objetos. Estes métodos são *QueryInterface*, que permite que um cliente consulte quais interfaces um objeto suporta e *AddRef* e *Release*, os quais gerenciam a contagem de referências para os objetos [FATOOHI1997].

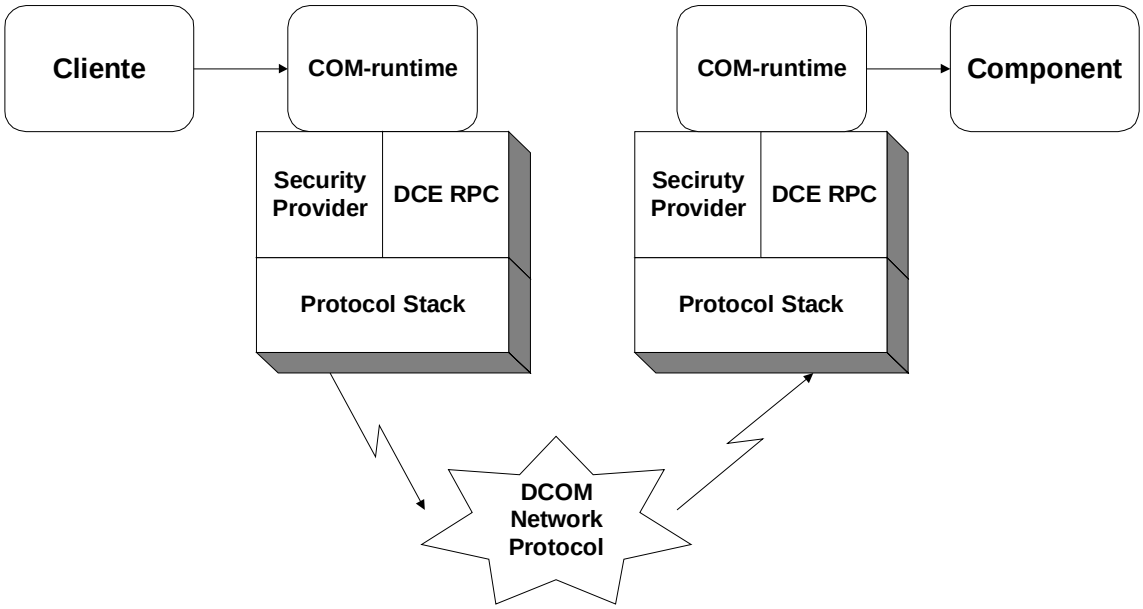


Figura 3.2: Arquitetura DCOM.

O DCOM é a extensão distribuída do COM, que constrói uma camada de chamada de procedimento de objetos remotos (*Object Remote Procedure Call* - ORPC) no topo do DCE RPC (*Distributed Computing Environment Remote Procedure Call*) para suportar objetos remotos (Figura 3.2). O COM define como os componentes de *software* e os seus clientes (outros componentes que se utilizam os seus serviços) interagem. Esta interação é definida tal que o cliente e o componente possam se conectar sem a necessidade de qualquer recurso intermediário. Um servidor COM pode criar instâncias de objetos de múltiplas classes de objetos. Um objeto COM pode suportar múltiplas interfaces, cada uma representando uma visão ou comportamento diferente do objeto. Uma interface consiste de um conjunto de métodos funcionalmente relacionados, e da mesma forma que a maioria dos sistemas baseados em objetos, o DCOM utiliza uma linguagem de definição das interfaces para os seus objetos.

O DCOM define um padrão binário por plataforma sem a preocupação de definir ligações com linguagens de alto nível, assim, os clientes e os desenvolvedores podem integrar componentes gerados com ferramentas de diferentes fabricantes e usá-los com diferentes implementações *runtime* do DCOM. Com o DCOM é possível distribuir uma única versão binária de um componente para uma dada plataforma que funciona com todos os outros

componentes e bibliotecas em tempo de execução. Qualquer linguagem que entenda este padrão pode criar e utilizar objetos DCOM. E com a distribuição do DCOM em todos os ambientes de desenvolvimento da Microsoft: Visual C++, Visual Basic, Visual J++ e Delphi/Pascal entre outras, ocasionou um aumento do número de linguagens e ferramentas que suportam esta tecnologia.

Com o DCOM, os detalhes sobre as questões de localização dos componentes não são especificados no código fonte, estando estes detalhes no mesmo processo ou em uma máquina em qualquer parte do mundo. Em todos os casos, a forma como o cliente se conecta a um componente e chama os métodos do componente é idêntica. Da mesma forma que o DCOM não requer mudanças no código fonte, também não é necessária a recompilação do mesmo. A independência de localização do DCOM simplifica a tarefa de distribuição dos componentes de uma aplicação. No caso de uma aplicação possuir numerosos pequenos componentes, a carga na rede pode ser reduzida caso estes componentes venham a se localizar em um mesmo segmento de uma rede local, em uma mesma máquina ou em outra localização específica. Por outro lado, se a aplicação for composta de um menor número de componentes maiores, estes poderão ser colocados em máquinas mais rápidas, sem a preocupação com a sua localização [DCOM1996].

Segundo Roque [ROQUE2000] um objeto DCOM é identificado por um identificador único de 128 bits (CLSID - *Class Identifier*), que associa uma classe de objetos a uma DLL (*Dynamic Linked Libraries*) ou aplicação executável (EXE) no sistema de arquivos. A utilização destes identificadores impossibilita a colisão de nomes entre classes, uma vez que estes não se encontram ligados aos nomes utilizados nos níveis inferiores da implementação. Por exemplo, pode-se escrever classes diferentes com o mesmo nome, mas cada uma delas terá um identificador diferente, dificultando as possibilidades de colisão.

Sempre que um cliente deseja criar uma instância de uma classe DCOM e utilizar os seus serviços, ele consulta o registro da máquina servidora que armazena o registro de todos os identificadores e a respectiva localização da DLL ou EXE nos servidores instalados no sistema. Assim, o cliente precisa apenas conhecer o identificador do arquivo que ele quer

acessar. Se este identificador não for encontrado no registro da máquina local, o DCOM irá fazer uma busca na rede para encontrá-lo.

3.1.3 JavaRMI

A JavaRMI é uma das abordagens da tecnologia Java para prover as funcionalidades de uma plataforma de objetos distribuídos. Através da utilização da arquitetura RMI, é possível que um objeto ativo em uma máquina virtual Java possa interagir com objetos de outras máquinas virtuais Java, independentemente da localização dessas máquinas virtuais. Ela tira partido do ambiente de operação Java na construção de aplicações cliente/servidor e sua arquitetura é dependente da linguagem Java.

A JavaRMI define que os objetos remotos são aqueles cujos métodos podem ser acessados a partir de outra JVM (*Java Virtual Machine*) que pode estar localizada em qualquer computador na rede. A interação dos clientes JavaRMI com os objetos remotos é feita através das suas interfaces de serviços que são oferecidos pelo objeto servidor, sem haver interação direta com as classes que implementam as interfaces. A sintaxe para invocação dos objetos remotos é a mesma para invocação de objetos locais. Os serviços especificados pelas interfaces RMI deverão ser implementados através de uma classe Java. Nessa implementação é preciso indicar que objetos dessa classe poderão ser acessados remotamente. Com a interface estabelecida e o serviço implementado, é possível criar as aplicações cliente e servidor RMI.

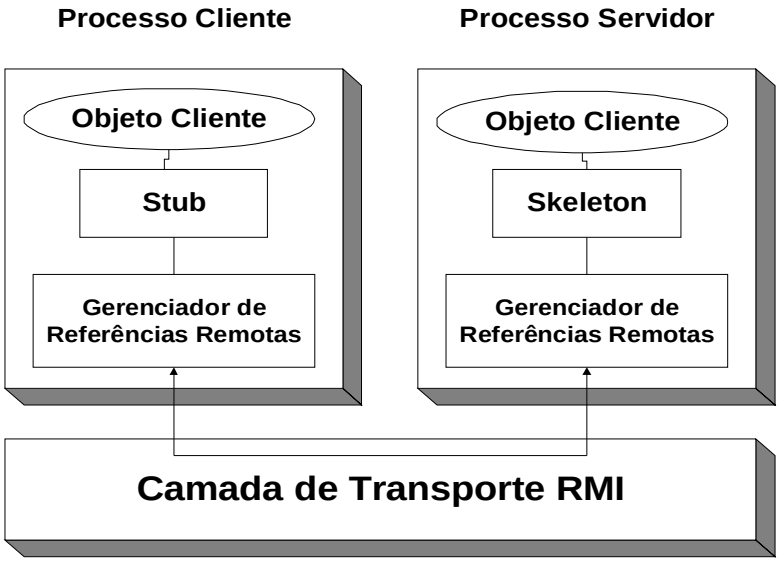


Figura 3.3: Arquitetura JavaRMI.

A arquitetura RMI oferece a transparência de localização através da organização de três camadas entre os objetos cliente e servidor (Figura 3.3) [FATOOHI1997]:

- A camada de *stub/skeleton*
- A camada de referência remota
- A camada do protocolo de transporte

A camada de *stub/skeleton* é a interface entre a camada de aplicação e a camada de referência remota. Ela oferece a interface que os objetos da aplicação usam para interagir entre si. A transmissão de dados desta camada para a camada mais baixa é feita através de um mecanismo chamado Serialização de Objetos que permite a transmissão transparente de objetos através das JVMs. A serialização de objetos suporta a codificação dos objetos em um fluxo de *bytes*. Para os objetos não remotos os parâmetros são passados por valor, já que eles são visíveis dentro de uma única JVM, mas para os objetos remotos, os parâmetros são passados por referência.

Na camada de referência remota são criadas e gerenciadas as comunicações entre os clientes e servidores nas JVMs para os objetos remotos, pois ela é o *middleware* entre a camada de *stub/skeleton* e o protocolo de transporte. A JavaRMI suporta as referências

persistentes bem como as referências não persistentes, além disso ela fornece meios diferentes de invocação tais como a invocação simples a um único objeto e a invocação a um objeto replicado em múltiplas posições, pois diferentes mecanismos de suporte são implementados nesta camada. Observa-se também que ela fornece estratégias para reconexão se um objeto remoto se tornar inacessível.

A camada de protocolo de transporte é responsável pela comunicação entre clientes e servidores, oferecendo o protocolo de dados binários que envia as solicitações aos objetos remotos pela rede. Ela ajusta, controla e monitora as conexões. A atual implementação do transporte é baseada em TCP, mas este pode ser substituído por um transporte baseado no UDP.

A JavaRMI estende algumas das características de Java permitindo novas operações, como as propriedades de segurança que foram ampliadas a fim de incluir o *downloading* de *stubs*. Ela adiciona algumas novas necessidades estruturais para o servidor e deixa a comunicação entre as JVMs transparente para o cliente, devido à sua integração à tecnologia de objetos distribuídos com a linguagem Java.

3.2 Características das Plataformas Analisadas

Com base nos conceitos apresentados sobre as plataformas CORBA, DCOM e JavaRMI, e na definição dos *Web Services* apresentada no capítulo 2, analisamos algumas características destas plataformas, demonstrando também como estas características se apresentam em relação aos *Web Services*. São elas: Suporte de Linguagens, Suporte de Plataformas, Modelo de Comunicação, Transações, Mensagens, Segurança, Localização, Tolerância a Falhas, Manutenção do Estado dos Objetos, Balanceamento de Carga, Coleta de Lixo Distribuída, Orientação a Objetos e Linguagem de Definição de Interface.

3.2.1 Suporte de Linguagens

Durante as fases de projeto e de implementação de uma aplicação distribuída enfrenta-se obstáculos para escolher a linguagem ou ferramenta a ser utilizada. A definição desta

linguagem leva em conta fatores como o custo do desenvolvimento, o conhecimento disponível e o desempenho. Um aspecto fundamental que deve ser considerado é o suporte das linguagens pelas diferentes plataformas no caso de uma interação com sistemas legados.

CORBA

A CORBA suporta várias linguagens dentro de uma única aplicação distribuída e provê alto grau de interoperabilidade, já que ela foi especificada para ser independente da linguagem e de plataforma. Desta forma, asseguramos que os objetos distribuídos construídos em diferentes produtos CORBA podem se comunicar.

Desde que as linguagens utilizadas para a implementação do sistema distribuído tenham implementações CORBA escritas para elas, permite-se que sistemas escritos nestas linguagens possam se conectar. Isto é possível porque a CORBA separa a interface da implementação e fornece dados neutros para que seja possível a chamada de objetos através de diferentes linguagens. A especificação define mapeamentos para linguagens como C, C++, Smalltalk, Ada, Java, OLE (Visual Basic, PowerBuilder, Delphi) e COBOL [CORBA, HOQUE1998].

DCOM

O DCOM é inteiramente independente da linguagem, já que é uma extensão do COM. Em princípio qualquer linguagem pode ser usada para criar componentes COM. Java, Microsoft Visual C++, Microsoft Visual Basic, Delphi, PowerBuilder e Micro Focos COBOL, todas interagem bem com o DCOM.

Devido a independência da linguagem DCOM, os desenvolvedores de aplicações podem escolher as ferramentas e as linguagens que são mais familiares para eles. Em virtude desta independência também torna possível uma rápida prototipagem onde os componentes podem ser desenvolvidos em uma linguagem de alto nível, como o Microsoft Visual Basic e o código ser reescrito mais tarde em uma linguagem diferente, como o C++ ou Java, podendo melhorar os benefícios com as características avançadas.

A garantia da portabilidade no DCOM está na forma de tradução das linguagens para o código binário. Devido às grandes diferenças de técnicas de tradução das linguagens, o DCOM tem como desvantagem a dificuldade para manter a compatibilidade a um nível tão baixo do *hardware* e isto cria vulnerabilidades devido inclusive ao progresso do próprio *hardware*. [6]

JavaRMI

A JavaRMI é uma tecnologia Java e esta é a única linguagem suportada diretamente por esta plataforma. Para ultrapassar esta limitação a JavaRMI disponibiliza uma API chamada JNI (*Java Native Interface*) que permite ao código Java chamar ou ser chamado por rotinas implementadas em outras linguagens. Os desenvolvedores utilizam o JNI para escrever os métodos nativos de Java e suportar as situações onde uma aplicação não pode ser escrita inteiramente em Java. Por exemplo, quando a biblioteca padrão de classes de Java não suporta as características de plataformas dependentes necessárias para a aplicação, ou já se tem uma biblioteca escrita em uma outra linguagem e o desejo de torná-la acessível ao código de Java, ou se quer executar em conjunto uma parcela pequena de código em uma linguagem de baixo nível. O inconveniente em recorrer a esta solução é o aumento significativo da complexidade das aplicações resultante da utilização das várias linguagens [JNI].

Web Services

Um *Web Service* é uma interface que descreve um conjunto de operações usando mensagens no padrão XML, como esta interface esconde os detalhes da implementação do serviço, ela permite que este seja usado independentemente da linguagem de programação. Nesta arquitetura os objetos da linguagem de programação são convertidos em uma mensagem XML e transportados através do protocolo SOAP. Quando chega ao destino a mensagem é novamente convertida para o formato requisitado pela aplicação para ser processada. O mesmo acontece na comunicação em sentido contrário na direção da aplicação servidora para a aplicação cliente. Esta conversão é governada pelos esquemas de codificação encontrados dentro da mensagem. Desta forma torna-se possível a utilização de

qualquer linguagem de programação para o processamento de informações [KREGER2001].

3.2.2 Suporte de Plataformas

A integração de plataformas diferentes é feita normalmente por aplicações distribuídas entre clientes e servidores. A dificuldade em integrar estas múltiplas plataformas acontece devido às diferenças significativas em muitos aspectos destas plataformas como: as diferentes filosofias de interface do usuário, os diferentes serviços de sistemas e o conjunto de protocolos de rede disponíveis. As arquiteturas devem ser capazes de suportar as plataformas recentes e as plataformas antigas e quando necessário suportá-las em conjunto.

CORBA

Um dos aspectos centrais desta arquitetura foi sempre o suporte multiplataforma. A independência de plataforma fornece vantagens competitivas para empresas que utilizam seus sistemas na Internet, pois elas não terão que reconstruir seus sistemas de *hardware* e *software* na rede ou forçá-los a se submeterem a uma solução de um único fornecedor. A capacidade dos ORB's CORBA interoperar sem considerar o fornecedor de origem torna a CORBA/IIOP uma solução ideal para Internet, especialmente considerando o grande tamanho desta rede e os distintos *hardwares* e *softwares* distribuídos nela. A CORBA/IIOP permitirá que quase todos o sistemas em operação hoje sejam incorporados comparativamente com menos modificações, então uma variedade de fornecedores poderão trabalhar juntos. Existem atualmente ORBs CORBA para diversas plataformas diferentes. São suportados por esta arquitetura diversos sistemas operacionais como Windows, Mac, OS/2, vários UNIX, OS/400, MVS e outros [HOQUE1998].

DCOM

A arquitetura DCOM permite a integração de estruturas de plataformas neutras de desenvolvimento e de ambientes de máquinas virtuais, bem como a integração de componentes feitos sob encomenda com alta performance em plataformas otimizadas em uma única aplicação distribuída. Ela permite o uso de serviços para plataformas específicas

ou de otimizações, e é imparcial na escolha de algum estilo de serviços de sistema sobre outros. Ela está aberta para todas as soluções de plataformas de desenvolvimento.

O DCOM define um padrão binário por plataforma para que os usuários e desenvolvedores possam mesclar componentes gerados por ferramentas diferentes até mesmo da qual o objeto DCOM em execução fora desenvolvido. Até 1998 o DCOM esteve praticamente limitado às plataformas Microsoft Windows 95 e NT. Atualmente o DCOM além de continuar disponível para as plataformas Microsoft também se encontra disponível para plataformas como: Unix, Linux e Mac [DCOM1996].

JavaRMI

Observa-se hoje o crescimento cada vez mais dos fornecedores de *software* que anunciam o suporte da JVM pelas suas plataformas, devido a crescente popularidade da linguagem Java. Como uma peça do núcleo da plataforma de Java, a JavaRMI está disponível em qualquer lugar que a plataforma estiver. Todas as plataformas com suporte da JVM estão aptas a poderem utilizar a JavaRMI. Atualmente além da Javasoft (SunOS e Solaris) e da Microsoft (Windows) já várias outras empresas anunciaram suporte a JVM para as suas plataformas, são elas: AIX, HPUX, Linux e MacOS.

Web Services

A interface do *Web Services* cobre todos os detalhes necessários para interagir com os serviços, incluindo os formatos da mensagem, os detalhes das operações, os protocolos do transporte e a localização em diversas plataformas. O suporte a estas plataformas ocorre através do uso do SOAP que é um protocolo que tem como característica executar a transferência de dados entre funções independente do ambiente de execução. Este protocolo define um mecanismo simples para expressar a semântica da aplicação fornecendo um modelo de empacotamento modular e codificação dos dados dentro dos módulos. Este fato o torna independente da plataforma de *hardware* ou do *software* em que é executado [GUNZER2002].

3.2.3 Modelo de Comunicação

A integração das diferentes tecnologias existentes ocorre através de mecanismos de comunicação com suporte robusto e diversificado nas plataformas de *middleware*. Para que esta integração seja possível, estas plataformas deverão ser transparentes em relação ao protocolo de comunicação.

CORBA

O modelo de comunicação da arquitetura CORBA está baseado no GIOP (*General Inter-ORB Protocol*) que define o formato das mensagens e a representação comum dos dados para comunicação entre os ORB's. O conceito por trás do GIOP são as transações de ORB para ORB. Ele foi projetado para operar diretamente sobre qualquer protocolo de transporte confiável e orientado a conexão. Como o OMG adotou o TCP/IP (*Transfer Control Protocol/Internet Protocol*) como protocolo de transporte padrão, surgiu então o IIOP (*Internet Inter-ORB Protocol*) que passa pedidos ou recebe respostas com a camada de transporte Internet TCP, ou seja, ele especifica como as mensagens GIOP são trocadas sobre a rede TCP/IP. O IIOP possibilita o uso da própria Internet como *backbone* entre os ORB's, ele provê interoperabilidade também entre ORB's compatíveis com o padrão CORBA. Ele especifica como as mensagens GIOP são trocadas sobre a rede TCP/IP [CORBA, HOQUE1998].

DCOM

O protocolo DCOM é implementado facilmente nas plataformas nas quais o DCE RPC já está disponível, pois ele é baseado na especificação DCE RPC da OSF (*Open Software Foundation's*) com algumas extensões. O DCE RPC define um padrão convertendo na memória os parâmetros dos pacotes da rede em estruturas de dados. Ele fornece um conjunto de tipos de dados portáveis.

O DCE RPC é um protocolo que define como são feitas as chamadas para um objeto e como são apresentadas, comunicadas e mantidas as referências dos objetos. Numa fase

inicial o protocolo utilizado pelo DCOM foi o UDP (*User Datagram Protocol*), sendo atualmente utilizado o protocolo TCP.

Os mecanismos de segurança do DCOM permitem a integração com ambientes de segurança baseados no DCE RPC. O Windows NT 4.0 pode servir como um *gateway* entre as plataformas que suportam RPC otimizado e as plataformas que fornecem somente suporte ao DCE RPC padrão [DCOM1996].

JavaRMI

O modelo de comunicações da arquitetura JavaRMI está baseado no protocolo nativo JRMP (*Java Remote Method Protocol*) que está na sua segunda versão com otimização em termos de performance no Java 2 SDK. Ela também possui o protocolo JavaRMI-IIOP lançado pela Sun em conjunto com a IBM para suportar o padrão CORBA. O JavaRMI sobre o IIOP é uma combinação das características da tecnologia da JavaRMI com as características da tecnologia CORBA. O RMI sobre IIOP é baseado em padrões abertos definidos com a participação de centenas de fornecedores e de usuários da OMG e fornece flexibilidade permitindo que os desenvolvedores passem qualquer objeto serializado do Java entre os componentes da aplicação. Ele usa o IIOP como seu protocolo de comunicação. O IIOP facilita a integração entre aplicações legadas e a plataforma Java permitindo que os componentes de aplicações escritos em C++, *Smalltalk* e em outras linguagens suportadas por CORBA se comuniquem com os componentes que executam nesta plataforma.

Com o RMI sobre o IIOP, pode-se escrever interfaces remotas e implementá-las apenas usando a tecnologia Java e a API RMI. Desde que a plataforma suporte o ORB para uma determinada linguagem que seja suportada pelo OMG, estas interfaces poderão ser implementadas nesta linguagem. De forma semelhante, os clientes podem ser escritos em outras linguagens usando uma IDL derivada das interfaces baseadas na tecnologia Java remota. Usando o RMI sobre IIOP, os objetos podem ser passados por referência e por valor para o IIOP [JRMI].

Web Services

O protocolo utilizado para os *Web Services* é o SOAP que permite a troca de informações em um ambiente descentralizado e distribuído. Ele facilita a publicação, localização, ligação e invocação das operações. O SOAP não se importa com o sistema operacional, a linguagem de programação ou o modelo do objeto que está sendo usado no lado do servidor ou no lado do cliente, este se comunica com os sistemas usando o protocolo XML baseado em texto, em vez de um formato binário usado pelos outros protocolos, isto o torna altamente interoperável por meio de várias plataformas. Ele pode ser transportado pelo SMTP, FTP e pelo HTTP onde é mais utilizado [GUNZER2002].

3.2.4 Transações

Nos últimos anos, este aspecto tem sido o centro das atenções das plataformas de *middleware*, pois com a descentralização do processamento surgiu a necessidade de implementar técnicas que garantissem a integridade das informações processadas neste novo ambiente. Uma das técnicas que possui esta característica é a transação, baseada no conceito de ação atômica. Esta técnica permite o encapsulamento de um conjunto de operações quando da sua execução de tal forma que, se o processamento ocorrer sem a presença de falhas, garante-se que todas as operações sejam executadas. Caso ocorra uma falha na execução de qualquer uma das operações do conjunto, garante-se que qualquer resultado obtido por este conjunto será ignorado, voltando o processamento ao estado anterior ao da execução da transação.

CORBA

A norma do serviço de transações da arquitetura CORBA especifica um vasto leque de serviços para suporte a transações distribuídas. A especificação do serviço de transação define que o serviço de transação deve prover um conjunto de operações definidas. Primeiro, o serviço necessita controlar o escopo e o tamanho da transação. Ele deve também permitir que os múltiplos objetos tenham envolvimento em uma transação atômica. O serviço deve permitir também aos objetos associar a transação com alguma mudança em seu estado interno e o serviço deve ajudar na coordenação da conclusão da

transação. Ele também provê uma boa camada de manipulação de erro e recuperação e pode ser valioso no provimento de integridade para o sistema [HOQUE1998].

DCOM

Neste aspecto a arquitetura DCOM também é uma ferramenta que implementa transações através da utilização do MTS (*Microsoft Transaction Server*). Cada aplicação baseada no MTS é constituída de componentes COM e os clientes podem acessar estas aplicações remotamente através do DCOM. O MTS fornece o suporte para a escalabilidade e a segurança nas transações.

JavaRMI

A JavaRMI utiliza um serviço de transações baseado na arquitetura CORBA. O serviço de transações de Java JTS (*Java Transaction Server*) especifica a execução de um gerente de transação que suporta a especificação JTA (*Java Transaction API*) em alto nível e implementa o mapeamento Java da especificação OMG OTS (*Object Transaction Service*) em baixo nível. O JTS usa as interfaces do CORBA OTS para a interoperabilidade e a portabilidade. Estas interfaces definem um mecanismo padrão para toda a implementação que utilizar o IIOP para gerar e propagar o contexto da transação entre os gerentes de transação de JTS. Isto permite também o uso de outra API sobre o mecanismo do transporte do IIOP. O gerente de transação JTS fornece os serviços de transação para as partes envolvidas nas transações distribuídas: o servidor da aplicação, o gerente de recursos, a aplicação transacional ***standalone*** e o gerente de recursos de comunicação [JTS].

Web Services

Atualmente não existe um padrão disponível capaz de definir transações para *Web Services*. Entretanto está surgindo um padrão definido pelo grupo OASIS (*Organization for the Advancement of Structured Information Standards*) que é o BTP (*Business Transaction Protocol*). Ele tem o objetivo de resolver problemas nos ambientes com interações complexas e com comunicações inseguras. O objetivo é fornecer uma conclusão ou um cancelamento concreto sob as regras do negócio que não necessitam e não possam ser compreendidas por todos os participantes da transação. Ele define mensagens de XML que

podem ser trocadas sobre muitos protocolos, incluindo a combinação de SOAP/HTTP. Fornecer a coordenação da transação para *Web Services* é uma das exigências para o BTP [BTPP2002].

3.2.5 Mensagens

As mensagens são objetos de dados cuja estrutura e aplicação são definidas pelos próprios sistemas que as utilizarão, sendo que a troca de mensagens é feita através de primitivas explicitas de comunicação que enviam e recebem as mesmas. Esta transmissão e recepção das mensagens de forma confiável tem que ser uma garantia das plataformas de *middleware*.

CORBA

A arquitetura CORBA delega ao serviço de eventos a tarefa de troca de mensagens. Este serviço provê a capacidade para sua aplicação CORBA enviar e receber mensagens. Baseado na arquitetura, pode-se decidir qual a manipulação de erro pode ser usada para executar as necessidades da aplicação. O serviço de eventos provê um canal de comunicação desacoplado entre os objetos CORBA que enviam e recebem mensagens com notificações de uma mudança em algum lugar do sistema. Ele é análogo ao modelo de eventos usado na maioria das interfaces de *software*, mas é útil em todos os tipos de programas. Nos programas CORBA, um dado servidor pode estar executando um processo, mas pode estar interessado em eventos gerados por outro servidor em algum lugar do sistema, então o serviço de eventos provê a capacidade de gerar e processar as requisições para obtenção destes serviços. O modelo de mensagens da especificação da arquitetura CORBA tem suporte para comunicações assíncronas [HOQUE1998].

DCOM

O DCOM fornece uma comunicação síncrona usando chamadas remotas de procedimento (*Remote Procedure Call* - RPC). Mas muitas aplicações distribuídas necessitam de uma comunicação assíncrona e o DCOM não suporta diretamente este tipo de comunicação, no entanto este problema foi solucionado pela Microsoft recorrendo a um mecanismo

complementar designado MSMQ (*Microsoft Message Queue*). O MSMQ fornece exatamente este tipo do serviço no Windows NT Server. Com o MSMQ, uma aplicação pode emitir mensagens para outra aplicação sem esperar uma resposta. As mensagens são emitidas para uma fila, onde são armazenadas até que uma aplicação de recepção as remova. Se uma resposta é esperada, o remetente pode verificar a fila de mensagens no tempo livre. A fila de mensagens é flexível e uma solução confiável para comunicação, sendo apropriada para muitos tipos das aplicações. O fato deste mecanismo não ser parte integrante do DCOM e estar limitado quase exclusivamente às plataformas Intel, faz com que o DCOM tenha algumas limitações no que diz respeito à interoperabilidade entre plataformas diferentes [DCOM1996].

JavaRMI

O JavaRMI recorre ao JMS (*Java Messaging Service*) para a implementação do serviço de mensagens. O serviço de mensagem do Java é uma API Java que permite que as aplicações criem, enviem, recebam e leiam as mensagens. A JMS API define um conjunto comum de interfaces e uma semântica associada que permitem que os programas escritos na linguagem de programação Java se comuniquem com outras implementações de sistemas de mensagens. Esta fornece características suficientes para suportar aplicações de mensagens sofisticadas. Ela também se empenha em maximizar a portabilidade de aplicações da JMS através dos fornecedores da JMS no mesmo domínio do sistema de mensagens.

A JMS API fornece uma comunicação assíncrona, onde um fornecedor de JMS pode entregar mensagens aos clientes enquanto elas chegam e o cliente não tem que pedir as mensagens para recebê-las. Esta permite que a comunicação seja acoplada frouxamente, fornecendo uma comunicação confiável, onde a JMS API pode assegurar que uma mensagem seja entregue uma única vez. Os níveis mais baixos de confiabilidade estão disponíveis para as aplicações que podem fornecer recursos para falha na entrega de mensagens ou receber mensagens duplicadas [HAASE].

Web Services

Não existe ainda garantia quanto a entrega da mensagem. O SOAP não tem suporte para mensagens confiáveis. Quando uma mensagem estiver sendo transferida, se o sistema falhar, o sistema não saberá como reenviar a mensagem. Uma solução para atenuar este problema está sendo desenvolvida pela IBM implementando mensagens confiáveis na camada de transporte através de uma extensão para o HTTP, o HTTP-R (*Hypertext Transfer Protocol Reliabl*e) que dirige as deficiências do protocolo propondo regras que tornem possível assegurar de que todas as mensagens sejam entregues no seu destino em seu formato exato e uma única vez. Nos casos onde a entrega da mensagem falha, o protocolo relatará a mensagem como não entregue. A mensagem SOAP transportada sobre o HTTP-R terá o mesmo formato da mensagem SOAP sobre o HTTP. A necessidade de informação adicional para correlacionar a requisição e a resposta no ambiente do HTTP-R assíncrono é colocada no cabeçalho da mensagem [TODD2002].

3.2.6 Segurança

A segurança é atualmente considerada como um fator decisivo no sucesso dos sistemas de objetos distribuídos, dado que um número crescente destas aplicações fornecem serviços fundamentais para o funcionamento normal das organizações. Sem suporte de segurança da plataforma de desenvolvimento distribuída, cada aplicação seria forçada a executar seus próprios mecanismos da segurança.

CORBA

O serviço de segurança é projetado de forma que possa prover uma política de segurança consistente entre os domínios dos objetos residentes em diferentes ORBs e em diferentes máquinas. Ele provê controle de acesso em níveis variáveis no sistema e faz provisões para uma comunicação segura do objeto através de uma rede insegura. Este serviço considera aspectos como integridade, responsabilidade, disponibilidade, confidencialidade e não-repúdio. Ele define as seguintes funcionalidades de segurança: controle de acessos, delegação, auditoria, autenticação e políticas de implementação.

DCOM

O DCOM usa a estrutura de segurança fornecida pelo Windows NT. Este fornece um conjunto de serviços internos de segurança que suporta múltiplos mecanismos de identificação e de autenticação. Uma parte central da estrutura da segurança é um diretório de usuário que armazena a informação necessária para validar credenciais de um usuário (nome do usuário, senha, chave pública). A maioria das implementações de DCOM em plataformas não Windows NT fornece um mecanismo similar ou idêntico de extensibilidade para usar o tipo de segurança disponível nesta plataforma. A maioria das implementações UNIX de DCOM incluirão um provedor de segurança compatível com o Windows NT.

O modelo de programação DCOM esconde a localização e os requisitos de segurança de um componente, podendo construir aplicações distribuídas seguras sem nenhum código de segurança específico ou projetá-las em cada cliente ou componente. O mesmo código binário que trabalha em um ambiente de uma única máquina, onde a segurança pode não ser necessária, pode ser usado em um ambiente distribuído de forma segura. O DCOM consegue esta transparência deixando que os desenvolvedores e os administradores configurem os ajustes de segurança para cada componente. O sistema de arquivos do Windows NT deixa os administradores ajustarem as listas de controle de acesso (ACLs) para arquivos e diretórios e o DCOM armazena as listas do controle de acesso por componentes. Estas listas indicam simplesmente que usuários ou grupos de usuários têm a direito de acessar um componente de alguma classe. Estas listas podem facilmente ser configuradas usando a ferramenta de configuração do DCOM (DCOMCNFG) ou através de programação usando o registro Windows NT e as funções de segurança de Win32. Desta forma a segurança DCOM está dependente do sistema operacional Windows NT [DCOM1996].

JavaRMI

A versão atual do RMI estende as diretivas de segurança do Java visando contornar os possíveis problemas ocasionados pelo *download* de *stubs*. A segurança no Java é fornecida através da classe *RMISecurityManager*, que faz a análise de todas as ações sensíveis de

segurança, tais como aberturas de arquivos e conexões de rede. Esta classe é tão restritiva quanto aquelas usadas para os *applets* (nenhum acesso a arquivo e assim por diante). Isto impedirá implementações de *download* dos dados de leitura ou de escrita do computador, ou conexões a outros sistemas ocultos sob o *firewall*. A classe *RMISecurityManager* é a responsável por aceitar conexões, escutar qualquer porta, manipular *threads* fora do seu grupo de processos, criar leitores de classes, usar DLLs, sair de uma JVM, fazer acesso a propriedades e abrir arquivos descritores. Caso esta política de segurança não seja suficiente podemos estendê-la criando uma nova classe de segurança derivada de *RMISecurityManager* e reforçar com diferentes restrições de segurança. As diretivas de segurança podem ser colocadas em alerta desde a primeira linha do programa Java, mas todos os comandos subseqüentes serão submetidos à política descrita acima [FRMI].

Web Services

Os mecanismos de segurança dos *Web Services* ainda são imaturos. O SOAP não define um mecanismo para autenticação de uma mensagem antes que ela seja processada. Como solução é implementada a segurança nos canais de transmissão usados para trocar as mensagens SOAP, utilizando o HTTP com o SSL (*Secure Socket Layer*) ou as VPNs (*Virtual Private Networks*). Também estão sendo desenvolvidos alguns mecanismos para manter a segurança na camada de mensagens como a assinatura digital em XML, a criptografia em XML e a SAML (*Security Assertion Markup Language*) que é uma linguagem comum para compartilhamento de serviços de segurança.

A assinatura digital em XML é um padrão seguro para verificar as origens das mensagens. A especificação da assinatura em XML permite que os documentos XML sejam assinados de uma forma padrão, com uma variedade de algoritmos de assinatura digitais diferentes. As assinaturas digitais podem ser usadas para validação das mensagens e para o não repúdio.

A Criptografia XML permitirá a codificação do conteúdo digital como imagens GIF (*Graphic Interchange Format*) e SVG (*Scalable Vector Graphics*) ou trechos XML. Ela permite que parte de um documento XML seja codificado enquanto outras partes ficam

abertas ou que seja feita a supercodificação dos dados, isto é, codificar um documento XML quando alguns elementos já estão codificados.

A SAML (*Security Assertion Markup Language*) é o primeiro padrão da indústria para as transações seguras de comércio eletrônico que usam XML. O SAML está sendo desenvolvido para fornecer uma linguagem comum para compartilhar serviços de segurança entre as empresas engajadas no *business-to-business* e nas transações *business-to-consumer*. O SAML permite que as empresas troquem de forma segura autenticação, autorização e informações de perfil entre seus clientes e parceiros considerando seus sistemas de segurança ou suas plataformas de *e-commerce*. Como resultado a SAML promove a interoperabilidade entre os diferentes sistemas de segurança, fornecendo a estrutura para transações seguras do *e-business* através das fronteiras da empresa [HONDO2002].

Outros esforços também estão sendo feitos para aumentar a segurança dos *Web Services*, entre eles a Microsoft e a IBM se uniram e lançaram a especificação *WS-Security* (*Web Services Security Language*) que descreve mecanismos para aumentar a qualidade da proteção através da integridade, da confidencialidade e de uma autenticação única da mensagem. A *WS-Security* é flexível e foi projetada para ser usada como base para a construção de uma grande variedade de modelos de segurança incluindo PKI (*Public Key Infrastructure*), *Kerberos* e SSL. Especificamente a *WS-Security* fornece suporte para múltiplos avisos de segurança, múltiplos domínios de confiança, múltiplos formatos de assinatura e múltiplas tecnologias de codificação [ATKINSON2002].

Um fator importante que deve ser considerado para a utilização dos *Web Services* é que eles normalmente são transportados sobre o HTTP na porta 80 e com isso passam facilmente pelo *firewall* situado entre a rede interna de uma empresa e a internet. Ele monitora todo o tráfego de fora para dentro, e bloqueia qualquer tráfego que não esteja autorizado, mas a porta 80 usualmente é deixada com pouca proteção. Enquanto a CORBA e a JavaRMI são utilizadas em outras portas e têm o seu acesso dificultado, os *Web Services* têm a vantagem de passar livremente pelo *firewall*, mas isto também pode se configurar em uma

desvantagem se o *Web Service* não implementar nenhuma funcionalidade de segurança. Os *Web Services* simplificaram as comunicações, mas também alteraram algumas medidas de segurança existentes. Então, o desafio agora é definir um modelo de segurança que desmonstre como os dados podem ser transportados através da aplicação e da rede de acordo com os requisitos especificados pelo negócio sem expor os dados a um risco inadequado.

3.2.7 Localização

Os objetos distribuídos são armazenadores de códigos que executam uma determinada tarefa. Sua principal característica refere-se a sua localização, eles podem ser abrigados todos em uma única máquina, ou distribuídos em máquinas distintas de uma rede de computadores (LAN - *Local Area Network*, WAN - *Wide Area Network*, e Internet). Em conjunto, esses objetos são capazes de executar funções para um sistema. Desta forma, uma das características principais a ser levada em conta nas diferentes plataformas de *middleware* é a sua capacidade de registrar e manter a localização dos serviços.

CORBA

O Serviço de Nomes pode ser usado para designar os nomes logicamente para os objetos e encontrar estes objetos facilmente, baseado em seus nomes compostos. Este serviço de localização pode ser configurado em tempo de execução e pode ser útil na localização de objetos em um sistema distribuído complexo. Ele provê a capacidade para os objetos CORBA encontrarem outros objetos, utilizando uma convenção de nomes facilmente diferenciada. Com a ajuda do serviço de nomes, um objeto pode encontrar direto um outro objeto usando o reconhecimento de nomes. O serviço de nomes também habilita a associação com o nome lógico a um objeto em tempo de execução, provendo mais flexibilidade para configurar o objeto no sistema distribuído. Com o processo normal de localização, um objeto pode ter somente um nome, mas o serviço de nome habilita também a associação de mais de um nome lógico para um único objeto.

DCOM

Por não especificar detalhes de desenvolvimento no código fonte, o DCOM permite que seus componentes se comuniquem independente de localização. Um cliente DCOM desconhece a localização do servidor com quem está se comunicando, podendo estar na mesma máquina ou em qualquer outro lugar. Em ambos os casos a comunicação e as chamadas de métodos serão idênticas. O objetivo deste serviço é a abstração das diferenças entre os vários serviços de localização através da disponibilização de uma interface normalizada.

JavaRMI

A JavaRMI pode usar muitos serviços de localização diferentes. Ela possui um serviço de localização simples chamado RMI *registry*. Este registro do RMI funciona em cada máquina que possui o serviço remoto dos objetos e aceita consultas para serviços por default na porta 1099. Entre os outros serviços destaca-se também a JNDI (*Java Naming and Directory Interface*).

A JNDI é uma API que foi projetada especificamente para Java utilizando o seu modelo de objetos para fornecer funcionalidades de localização para as aplicações. A JNDI permite que as aplicações acessem diferentes serviços de localização, armazenem e recuperem objetos de qualquer tipo registrados no Java. Esta fornece métodos para uma implementação padrão de localização, tais como associar atributos aos objetos e procurar por estes objetos usando seus atributos. Além disto, a JNDI permite que as aplicações Java coexistam com sistemas legados e que diferentes serviços sejam conectados a uma API comum, permitindo que estas aplicações utilizem outros serviços de localização como LDAP (*Lightweight Directory Access Protocol*), NDS (*Novell Directory Service*), DNS (*Domain Name Service*) e NIS(YP) (*Network Information Service*) [JNDI].

Web Services

A publicação e a descoberta dos *Web Services* é feita através de um registro do serviço baseado no padrão UUDI. Uma vez que um *Web Service* é publicado, um requisitante do serviço pode encontrá-lo através da interface UDDI. O registro do UDDI fornece ao

requisitante uma descrição do serviço em WSDL e uma URL que aponta para o serviço. O requisitante pode então usar esta informação para se ligar diretamente ao serviço e invocá-lo.

3.2.8 Tolerância a Falhas

Tolerância a falhas é uma funcionalidade essencial para as aplicações de missão crítica que precisam de alta disponibilidade. Ela consiste na habilidade de um sistema responder a uma falha inesperada de *hardware* ou do *software*. Para isso, são utilizadas várias técnicas, dentre elas, a replicação dos sistemas de computação é uma das mais utilizadas. Como conseqüência, para obter esta alta disponibilidade é necessária a alocação de vários recursos de *hardware*, sistemas operacionais e de mecanismos de aplicação de *software*.

CORBA

O OMG publicou uma especificação para introduzir tolerância a falhas na arquitetura CORBA (FT-CORBA) [OMG2000]. Esta especificação procura fornecer um suporte robusto para as aplicações que requerem um nível elevado de confiabilidade, utilizando vários mecanismos como a replicação de objetos, técnicas de detecção e recuperação de falhas, além das definições para interoperabilidade próprias da arquitetura CORBA. Segundo Lung [LUNG2000] a especificação apresenta a definição de protocolos que podem ser separados em três módulos: Gerenciamento de Replicação (SGR); Gerenciamento de Falhas (SGF) e Gerenciamento de Recuperação e Logging (SLR). Sendo que o Serviço de Gerenciamento de Replicação é responsável pelo gerenciamento das propriedades de tolerância a falhas e do grupo de objetos, controlando as entradas e saídas dos objetos replicados; o Serviço de Gerenciamento de Falhas é responsável pela definição das interfaces dos serviços de detecção de falhas, notificação de falhas e análise de falhas; e o Serviço de Gerenciamento de Recuperação e *Logging* é responsável pelos mecanismos para transferência de estado de objeto e recuperação de réplicas faltosas.

DCOM

O DCOM fornece suporte básico para tolerância a falhas no nível de protocolo, utilizando um mecanismo simples para detectar as falhas baseando-se em um contador de referências. Este mecanismo restabelece as conexões automaticamente, caso a rede se recupere antes de um intervalo de tempo definido. Este contador verifica se a ligação objeto-servidor está ativa através do envio de mensagens em intervalos de tempo definido, no caso do servidor enviar um número de mensagens estabelecido no sistema (geralmente 3 mensagens) e não receber a resposta do cliente, ele termina a ligação e decrementa o contador de referência em uma unidade. Ao atingir-se o valor 0 no contador de referências significa que o servidor pode ser removido.

Algumas técnicas são utilizadas para recuperar as falhas no sistema, dentre elas, temos uma técnica que utiliza um componente de referência que gerencia as conexões, armazenando as informações sobre quais servidores estão mais disponíveis. No caso em que um cliente detecta uma falha, ele consultará este componente de referência e este fornecerá ao cliente automaticamente uma nova instância do componente em execução em uma outra máquina, porém as informações anteriores e sua consistência serão perdidas e a aplicação deverá tratar esta ocorrência em níveis superiores. Outra técnica utilizada é chamada de "***hot backup***". Nela tem-se duas cópias do mesmo componente servidor que são executadas em paralelo em diferentes máquinas, processando as mesmas informações. Os clientes podem se conectar a ambas as máquinas simultaneamente. Os componentes DCOM tornam esta ação completamente transparente à aplicação do cliente introduzindo código do servidor no lado cliente, que trata a tolerância a falha. Uma outra solução usaria um componente de coordenação que funciona em uma máquina separada, que emite os pedidos do cliente a ambos os componentes do servidor em nome do cliente [DCOM1996].

JavaRMI

A linguagem Java não oferece suporte a Tolerância a Falhas que permita a um serviço distribuído continuar funcionado corretamente caso alguns de seus componentes falhem. Quando ocorre uma falha na conexão no RMI, esta é tratada como uma ***RemoteException*** e a ligação ao servidor é perdida, pois não há religação automática. O que existe hoje são

soluções particulares propostas por diversos desenvolvedores espalhados pelo mundo que procuram suprir as necessidades de tolerância a falha [CIRNE2000, ZHONGWEI].

Web Services

Ainda não existe nenhum padrão para o desenvolvimento de tolerância a falhas em *Web Services*. Mas estudos estão sendo feitos por diversos fabricantes para elaborar uma especificação. Várias propostas estão sendo feitas com diferentes modelos, dentre eles, um modelo onde o *Web Service* deve implementar uma interface onde a arquitetura estenda dinamicamente esta interface no WSDL com métodos para tolerância a falhas; as aplicações que fazem uso de diferentes *Web Services* teriam que declarar suas interdependências, que serão usadas por um gerenciador de falhas para controlar a recuperação da falha e uma extensão da camada de comunicação SOAP poderá registrar e reproduzir as mensagens transmitidas [DIALANI2002].

3.2.9 Manutenção do Estado dos Objetos

A manutenção do estado do objeto se refere à habilidade dos objetos em manter o estado fora do escopo do cliente ou do objeto que originalmente o criou.

CORBA

O serviço de persistência dos objetos na CORBA provê funcionalidades de armazenamento e recuperação dos objetos de uma base de dados específica. O objeto pode ser salvo em algum tipo de base de dados e recuperado mais tarde quando for necessário. Esta base de dados pode ser qualquer mecanismo de armazenamento incluindo arquivos simples ou bancos de dados relacionais. Este serviço pode ser implementado de forma transparente para o cliente e este não precisa saber se o objeto está armazenado na memória ou no disco. Os clientes e objetos podem ter vários requisitos de persistência diferentes baseado no tipo da aplicação em que ele executa, podendo usar um protocolo de armazenamento específico se ele desejar.

DCOM

Um cliente DCOM não pode reconectar exatamente a mesma instância de objeto com o mesmo estado em tempos distintos. Ele pode somente reconectar a um ponteiro de interface da mesma classe. Consequentemente, os objetos DCOM não mantêm os estados entre as conexões. Observa-se também que no DCOM existe a necessidade de conservar o estado do objeto no cliente, já que uma perda da conexão resultará na perda do estado dos objetos. Isto significa que ele confia no cliente para fornecer a informação necessária para restaurar um objeto do armazenamento quando for necessário, o que pode causar um grande problema em ambientes onde existem conexões propensas a falha como a Internet.

JavaRMI

A JavaRMI mantém o estado dos objetos entre conexões através do serviço de ativação. A ativação do objeto é um mecanismo que provê referências persistentes aos objetos e gerencia a sua implementação. Na JavaRMI, a ativação permite que objetos iniciem sua execução a medida que esta ação seja necessária. Quando ativado, o objeto remoto é acessado, e se ele não estiver sendo executado, o sistema inicia a execução do objeto dentro da "*Java Virtual Machine*" apropriada.

Web Services

Não existe hoje nenhuma especificação em relação à manutenção do estado dos objetos nos *Web Services*, o que tem sido feito são iniciativas isoladas na busca de soluções para recuperar este estado na ocorrência de falhas. Estas iniciativas têm caminhado no sentido de desenvolver estratégias para criar objetos nos *Web Services* que guardam o seu estado. Elas propõem que as informações deste estado sejam serializadas e armazenadas em um objeto *Session*. Então a aplicação poderia recuperar estas informações quando recebesse cada nova requisição. Da mesma forma, todo o estado global dos objetos poderia ser serializado no contexto da aplicação [WAGNER2001].

3.2.10 Balanceamento de Carga

Com a ocorrência de diversos problemas de desempenho, incluindo tempos de resposta baixos, congestionamento da rede e interrupção de serviços, vem surgindo diversas

abordagens de como esses problemas podem ser contornados. O balanceamento de carga entre servidores é uma das técnicas utilizadas, consistindo na diversificação e no planejamento dos servidores, levando em consideração a quantidade de acesso por segundo e as aplicações que nele residem.

CORBA

Não existe hoje nenhuma especificação do OMG para balanceamento de carga na CORBA, o que se tem encontrado são diversas propostas que procuram suprir esta necessidade na arquitetura. Uma destas propostas [OTHMAN2001] apresenta um estudo sobre diversas estratégias de serviços de balanceamento de carga que podem ser executadas eficientemente usando as características do padrão CORBA. Além disso, várias implementações comerciais da CORBA também fornecem o serviço de balanceamento de carga como o IONA's ORBIX que pode executar o balanceamento usando o serviço de nomes da CORBA.

DCOM

O DCOM fornece algumas formas de implementação de tipos de balanceamento de carga. Entre eles tem-se o balanceamento de carga estático e o balanceamento de carga dinâmico. No balanceamento de carga estático o DCOM utiliza um servidor dedicado de referência em uma localização conhecida que armazena os nomes dos servidores onde o componente cliente recupera a informação toda vez que ele precisa se conectar a um servidor específico. O servidor de referência pode usar os mecanismos de segurança do DCOM para identificar o usuário requisitante e escolher o servidor dependendo de quem está realizando a requisição. Este pode também estabelecer uma conexão com o servidor e retorná-lo diretamente para o cliente, ao invés de retornar apenas o nome do servidor. No balanceamento de carga dinâmico o DCOM utiliza um servidor de referência com informações mais completas para fazer o balanceamento. Ao invés de direcionar o usuário com base em um identificador, ele analisa as informações sobre a carga do servidor, topologia de rede entre o cliente e os servidores disponíveis e estatísticas sobre a demanda passada de um dado usuário. A cada vez que um cliente se conecta a um componente, o

servidor de referência pode associá-lo para o servidor disponível mais adequado no momento [DCOM1996].

JavaRMI

Não foi encontrada nenhuma referência sobre este assunto para a especificação JavaRMI.

Web Services

O Balanceamento de carga para os *Web Services* está relacionado ao desempenho das máquinas na Web, porque na maioria das situações é o servidor Web que recebe a solicitação do SOAP pelo HTTP. Diversas técnicas podem ser utilizadas para fornecer este balanceamento, entre elas a utilização de replicação do *Web Service* nos vários servidores distribuídos na Internet que compartilham a carga adequadamente. Além disso, pode ser usado também um *cache* nos servidores de balanceamento de carga para que eles recuperem o resultado da execução dos *Web Services* a partir de seu armazenamento local, em vez de fazer chamadas pela Internet, acelerando consideravelmente o tempo de resposta.

3.2.11 Coleta de Lixo Distribuída

Em um sistema distribuído, deve haver um dispositivo coletor de lixo capaz de liberar automaticamente objetos remotos que não estejam sendo utilizados por nenhum cliente. O coletor de lixo deve gerenciar quais e quantos clientes ainda estão ativos, liberando corretamente as conexões inativas. Destaca-se também a importância da segurança da implementação deste serviço para evitar que um servidor remoto possa ser erroneamente liberado se uma ligação falhar, gerando problemas mais complexos.

CORBA

Não realiza coleta de lixo distribuída.

DCOM

O DCOM fornece um mecanismo de coleta de lixo distribuído robusto que é completamente transparente à aplicação, oferecendo um protocolo de rede simétrico com

uma interação unidirecional e uma comunicação interativa entre clientes e servidores. Este protocolo é utilizado para detectar se os clientes ainda estão ativos, através do monitoramento das máquinas clientes que emitem uma mensagem periódica. O DCOM considera uma conexão como quebrada se mais de três períodos de tempos definidos para troca de mensagens passarem sem o recebimento de nenhuma mensagem do componente. Se a conexão for considerada quebrada, o DCOM decrementa a contagem e libera as referências do componente, se a contagem alcançar zero o componente pode ser coletado como lixo.

JavaRMI

A plataforma Java oferece um coletor automático de lixo que recupera a memória de qualquer objeto rejeitado pelo programa que está executando. Na JavaRMI esta função é implementada pela referência remota que controla o tempo de conexão do cliente, assumindo que se ele se desconectar por um período de tempo específico, o objeto que estava conectado a ele pode ser coletado como lixo e uma exceção notificará o programa quando isto ocorrer. Este processo de remoção dos objetos remotamente não-referenciados ocorre de maneira automática.

Através da comunicação com o cliente, o servidor é notificado quando uma referência é liberada na aplicação remota, atualizando a sua lista de referências remotas para os objetos oferecidos por ele. Cada referência remota recebe também um período de validade; quando esse período expira, a referência é eliminada e o cliente é notificado. Esse mecanismo provê uma solução para os casos onde ocorrem falhas nos objetos referenciados por clientes e ficam impedidos de sinalizar que a referência foi liberada [FRMI].

Web Services

Não existe especificação para coleta de lixo distribuída.

3.2.12 Orientação a Objetos

A orientação a objetos trabalha com o conceito de relacionamento entre partes, sendo que cada parte é vista como um objeto que funciona separadamente e se relaciona com os

outros através da chamada de métodos. Em um ambiente distribuído estes objetos podem estar localizados em qualquer máquina de uma rede de computadores, já que eles possuem uma interface específica onde os compiladores geram um código específico para que outros objetos trabalhem com ele desconhecendo o local onde ele está abrigado.

CORBA

A CORBA é orientado a objetos encapsulando os objetos em uma arquitetura definida e fortemente integrada. Cada objeto tem uma interface definida que é utilizada pelos outros objetos para requisitarem seus serviços.

DCOM

O DCOM é orientado a objetos e possibilita a comunicação de objetos em máquinas e processos diferentes, onde o cliente descobre através da interface do DCOM qual o ClassID (CLSID) do objeto e solicita ao servidor a criação do objeto e o retorno do ponteiro para a interface do mesmo.

JavaRMI

A JavaRMI é orientada a objeto e seus objetos são acessíveis através de uma interface remota. Ela utiliza a linguagem Java e como esta é orientada a objetos, a JavaRMI não faz chamadas a procedimentos remotos, mas a métodos de um objeto localizado em uma máquina remota. O lado cliente deve possuir a interface (classe abstrata) do objeto remoto para poder referenciá-lo, enquanto que o lado servidor possui a implementação do mesmo.

Web Services

Os *Web Services* não são orientados a objetos e sim orientados a serviços. Eles não são vistos como componentes de um sistema e sim como módulos de *software* que oferecem serviços. A interface do serviço especifica apenas o que ele faz, não importando para o cliente como ele faz. O cliente requisita o serviço e apenas aguarda a resposta. Eles possuem a vantagem de serem orientados a mensagens e o desenvolvedor pode escolher entre usar a troca de mensagens usando RPC ou usando Mensagens. A troca de mensagens é feita através de um mecanismo de entrega que utiliza filas de mensagens assíncronas que

são entregues independentemente da aplicação destino estar ativada, e consequentemente a aplicação entrega o documento na fila e fica livre para executar outras tarefas.

3.2.13 Linguagem de Definição de Interfaces

Uma característica importante nos sistemas distribuídos é a possibilidade de comunicação entre as aplicações através de uma rede sem que necessariamente estas compartilhem a mesma linguagem. Isto é possível disponibilizando a descrição dos seus serviços através de interfaces para que as outras aplicações possam interagir e se adequar dinamicamente a estes serviços. Cada tecnologia implementa uma forma para que as informações sobre a implementação dos objetos que serão chamados fiquem disponíveis, isto normalmente é feito através de uma linguagem de definição de interfaces (IDL).

CORBA

Na CORBA a IDL fornece informações sobre os serviços que o objeto dispõe, como fazer para chamar estes serviços e o que devem esperar como resposta. Ela é a linguagem adotada para especificar a sintaxe de todas as interfaces, sendo que a IDL está desvinculada de qualquer informação referente a implementação do objeto, se apresentando apenas com declarações.

DCOM

Um objeto DCOM suporta um certo número de interfaces que especificam quais são os métodos disponíveis e como invocá-los. A linguagem que ele utiliza para definir as interfaces é MIDL (*Microsoft Interface Definition Language*) baseada no DCE. Ela define a assinatura do método, incluindo todos os tipos de dados, tipos de estruturas de dados e tamanhos de vetores que pertencem a lista de parâmetros do método.

JavaRMI

A JavaRMI não especifica nenhuma linguagem de definição de interface.

Web Services

A WSDL é o equivalente a IDL na CORBA. Ela descreve as interfaces dos serviços oferecidos por uma aplicação de forma que um cliente possa utilizar este serviço dinamicamente. Além de descrever a interface, a WSDL também contém a localização do serviço.

3.3 Tabela de Comparação das Plataformas

A Tabela 3.1 mostra um resumo da comparação das características das plataformas vistas anteriormente.

	CORBA	DCOM	JavaRMI	*Web Services*
Suporte de Linguagens	várias	várias	Java	várias
Suporte de Plataformas	várias	várias	várias	várias
Modelo de Comunicação	GIOP	DCE RPC	JRMP ou JRMI-IIOP	SOAP
Transações	Possui vários Serviços de Transações	Possui o MTS	Possui o JTS	Não existe ainda. No futuro pode ser o BTP
Mensagens	Utiliza o Serviço de Eventos	Utiliza o MSMQ	Utiliza o JMS	Poderá usar o HTTP-R
Segurança	Serviço de Segurança	Utiliza Serviço de Segurança do Windows NT	Utiliza a classe RMISecurityManager	Utiliza o SSL e o SAML
Localização	Serviço de Nomes	Serviço de Nomes	RMI registry e JNDI	UDDI
Tolerância a Falhas	FT-CORBA	Suporte no nível de protocolo	Não existe	Não existe
Manutenção do Estado dos Objetos	Serviço de Persistência	Não existe	Serviço de Ativação	Não existe
Balanceamento de Carga	Implementações comerciais	Estático e Dinâmico	Não existe	Utiliza a implementação dos servidores Web
Coleta de Lixo Distribuída	Não existe	Existe	Existe	Não existe
Orientação a Objetos	Sim	Sim	Sim	Não. Orientação a Serviços
Linguagem de Definição de Interfaces	IDL	MIDL	Não existe	WSDL

Tabela 3.1: Comparação das Plataformas.

3.4 Considerações Finais

Neste capítulo, apresentou-se uma comparação das principais características das tecnologias de sistemas distribuídos em conjunto com a sua respectiva representação na arquitetura dos *Web Services*. Diante deste cenário observamos alguns problemas que podem transformar em desvantagem a adoção desta tecnologia. Suas características devem ser avaliadas cuidadosamente em conjunto com os requisitos de seu sistema, verificando se a utilização dos *Web Services* atenderão eficientemente a estes requisitos.

Esta tecnologia ainda apresenta-se imatura não fornecendo suporte para propriedades de Qualidade de Serviço, preferências de usuários ou suporte a transações de forma padronizada. Além disso, ela não trabalha com os conceitos de Orientação a Objetos como herança e polimorfismo. A elaboração de um código para os *Web Services* requer um baixo nível de abstração, onde o programador tem que construir a mensagem, colocar os argumentos a serem passados na mensagem e em seguida enviá-la. Ele deve preocupar-se também com os nomes dos métodos, parâmetros utilizados e prover serialização e deserialização de alguns tipos de dados utilizados. Ela também não fornece suporte para construção e interoperabilidade de sistemas em tempo real. Outro problema é a dificuldade para debugar o código escrito, pois um erro somente será descoberto depois que a mensagem XML seja validada no servidor em tempo de execução e uma mensagem de erro seja retornada. Por fim, pode ocorrer falta de performance devido a necessidade de analisar e transportar o XML, pois apesar dos arquivos XML serem simples, nos casos de arquivos volumosos haverá grande consumo de memória e processamento na sua interpretação.

Entretanto, as suas vantagens não podem ser esquecidas, já que elas vêem suprir algumas falhas que existem nas outras tecnologias existentes hoje. O suporte a múltiplas linguagens de programação, a facilidade de transporte através dos *firewalls*, o seu projeto específico para o ambiente da Internet, e principalmente o investimento e o suporte dos grandes provedores de tecnologia, tornam os *Web Services* uma tecnologia promissora que está ganhando bastante espaço entre os desenvolvedores de *software*. Devido a suas falhas, eles não podem substituir as outras tecnologias, mas podem ser utilizados em conjunto com elas

suprindo algumas necessidades que as outras não oferecem de acordo com o contexto do sistema. Os *Web Services* podem ser utilizados como uma ponte de ligação entre os *middleware* dentro das empresas. Como exemplo, ele pode ser utilizado junto com a CORBA permitindo que os recursos desta tecnologia sejam distribuídos no ambiente de internet, já que ela apresenta algumas dificuldades para isso. O sistema pode ser construído com a CORBA utilizando seus recursos robustos e a interoperabilidade com os outros sistemas podem ser fornecidos pelos *Web Services* que terão maior facilidade de prover esta comunicação. Visto que muitas empresas utilizam mais que um tipo de *middleware*, alguns deles permitindo um alto nível de interoperabilidade, mas outros com dificuldade de manter esta característica, os *Web Services* são uma boa escolha para atuar na interconexão entre eles, desde que sejam observados os requisitos dos sistemas.

Com base nesta apresentação conclui-se que esta arquitetura ainda se encontra muito precoce, faltando muita coisa para ser desenvolvida. A sua proposta é inovadora e procura resolver muitos problemas de interoperabilidade, mas o desenvolvedor deve prestar atenção às suas limitações, observando as situações na qual ela pode ser utilizada para evitar a construção de sistemas complexos e outros problemas no atendimento de suas necessidades. Um dos principais aspectos que deve ser considerado é a utilização desta arquitetura em conjunto com outras mais robustas, de forma que os *Web Services* possam simplificar alguns processos e suprir algumas falhas existentes nestas outras tecnologias. Desta forma, eles poderiam atender um dos seus objetivos principais que é a integração de sistemas, procurando reduzir as complexidades inerentes às operações até que estes possam alcançar a maturidade mais rapidamente.

Capítulo 4

Estudo de Caso

A tecnologia de *Web Services* vem surgindo com a intenção de facilitar a integração de diversos sistemas heterogêneos. Ela está sendo apresentada como uma solução para os problemas de comunicação entre os *softwares*, porque apesar do avanço das outras tecnologias no sentido de oferecer aos desenvolvedores funcionalidades que antes deveriam ser implementadas por eles, a comunicação destes *softwares* com sistemas antigos e com outras soluções baseadas em múltiplos fabricantes e múltiplas tecnologias ainda é deficiente.

Diante deste cenário, onde dentro das empresas temos diferentes ambientes baseados em *mainframes* e sistemas distribuídos de computação, torna-se complicado o compartilhamento de informações e a integração das aplicações, fazendo com que os problemas com interoperabilidade assumam um papel muito importante. Dessa forma, os principais fornecedores do mercado de tecnologia indicam a tecnologia de *Web Services* para solucionar a questão de como fazer que uma aplicação possa usar funcionalidades de outras aplicações sem que ela tenha que ser reescrita.

Neste contexto também, as empresas de telecomunicações encontram dificuldade para integrar diversas aplicações com diferentes soluções para os problemas característicos desta área específica, na busca da conquista de uma boa aceitação para um mercado em expansão. Com base nesta situação temos como proposta o estudo desta nova tecnologia visando o aspecto da interoperabilidade na aplicação desta ao contexto da integração dos sistemas de gerência de redes de telecomunicação. Através de uma simulação estudamos a integração entre os sistemas CPqD Gerência de Planta, antigo SAGRE (Sistema Automatizado de Gerência de Rede Externa), e o CPqD Supervisão Remota, antigo SSR (Sistema de Supervisão Remota), onde avaliamos quais as vantagens e desvantagens do uso

desta tecnologia e apresentamos quais os aspectos que devem ser considerados para a utilização real deste protótipo.

4.1 Descrição do Projeto

O protótipo desenvolvido em nosso estudo sobre a aplicação da tecnologia de *Web Services* simula a integração entre os *softwares* CPqD Gerência de Planta e CPqD Supervisão Remota. Esta integração ocorre através da possibilidade do CPqD Gerência de Planta utilizar as informações na base de dados do CPqD Supervisão Remota, agregando novas funcionalidades para o gerenciamento da planta de telecomunicações como a possibilidade de criar novos mapas temáticos e facilitar o processo de implantação e manutenção da rede de TPCI (Telefones Públicos de Cartão Indutivo), através da obtenção de dados sobre os TPCI como a receita dos créditos, estado do aparelho, entre outras.

A integração *online* ocorre através da consulta direta do módulo de Telefonia Pública do CPqD Gerência de Planta à base de dados do CPqD Supervisão Remota, utilizando *Web Services*, evita a reescrita dos códigos e a replicação de dados na base do CPqD Gerência de Planta, tornando os sistemas mais flexíveis. A idéia do protótipo desenvolvido constitui de clientes web implementados no CPqD Gerência de Planta que podem consultar uma base de dados através de *Web Services* implementados no CPqD Supervisão Remota e recuperar dados relevantes para o processamento do CPqD Gerência de Planta. A seguir apresentamos de forma geral os sistemas CPqD Gerência de Planta e CPqD Supervisão Remota, e detalhamos o funcionamento do projeto.

4.1.1 CPqD Gerência de Planta

O CPqD Gerência de Planta é formado por um conjunto de aplicações que proporcionam o gerenciamento da planta de uma rede de telecomunicações, disponibilizando os dados geográficos em conjunto com as características técnicas dos elementos da rede externa. O sistema facilita o tratamento das informações representativas da situação real da rede

externa, oferecendo uma maior flexibilidade nas atividades de gerência e manutenção e no estudo do planejamento de expansão da rede de telecomunicação.

A rede externa é constituída pelo conjunto de equipamentos destinados à transmissão dos sinais telefônicos, como cabos, estações, postes, armários de distribuição e outros. O CPqD Gerência de Planta armazena as informações sobre estes equipamentos e fornece suporte para as diversas etapas do projeto, instalação e manutenção da rede. Ele é baseado em um banco de dados geográfico que relaciona os dados dos equipamentos, analisando-os no contexto da rede e os apresenta de forma distinta, dependendo do módulo do sistema utilizado.

O CPqD Gerência de Planta foi desenvolvido com uma arquitetura em três camadas, oferecendo: um desempenho otimizado, independentemente do número de usuários que operam o sistema; uma maior flexibilidade na arquitetura da rede na empresa, para a implantação do sistema; distribuição da carga de processamento sobre os servidores de aplicação; rápido desenvolvimento e/ou personalização das aplicações-cliente; independência dessas aplicações em relação às mudanças nas estratégias de armazenamento de dados; e facilidade no desenvolvimento de interfaces com outros sistemas.

Além disso, o *software* utiliza uma arquitetura aberta, com sistemas operacionais comerciais como UNIX e Windows. Ele é constituído por um conjunto integrado de módulos que consultam e manipulam os dados armazenados no banco de dados corporativo, garantindo o objetivo de automatizar o gerenciamento da rede. Seus módulos são: Cadastro , Engenharia, Operação, Celular, Interconexão, *Web Report*, *Web Viewer*, Administração, *Market* e Telefonia Pública. Entre eles, os módulos que mais interessam ao escopo deste trabalho são o módulo de Telefonia Pública e o módulo de *Market*. O módulo de *Market* permite o registro de informações de mercado e de perfil de consumo, com o objetivo de caracterizar o atendimento ao cliente e realizar um estudo de demanda. Oferece também a possibilidade de espacializar dados contidos em bancos de dados externos. Essas informações podem ser consultadas em forma de mapas temáticos que se tornam um recurso importante para a tomada de decisões. Já o módulo de Telefonia Pública trata do

gerenciamento dos telefones de utilização pública, cuja responsabilidade é da empresa de telecomunicação. Esse módulo foi desenvolvido baseado nas normas reguladoras da ANATEL (Agência Nacional de Telecomunicações) para melhorar o atendimento aos clientes. Com as funcionalidades geográficas do CPqD Gerência de Planta, ele realiza simulações para determinar a melhor localização para a instalação de novos telefones públicos [SASAOKA2002]. A Figura 4.1 mostra uma cópia da tela do módulo de Telefonia Pública do CPqD Gerência de Planta.

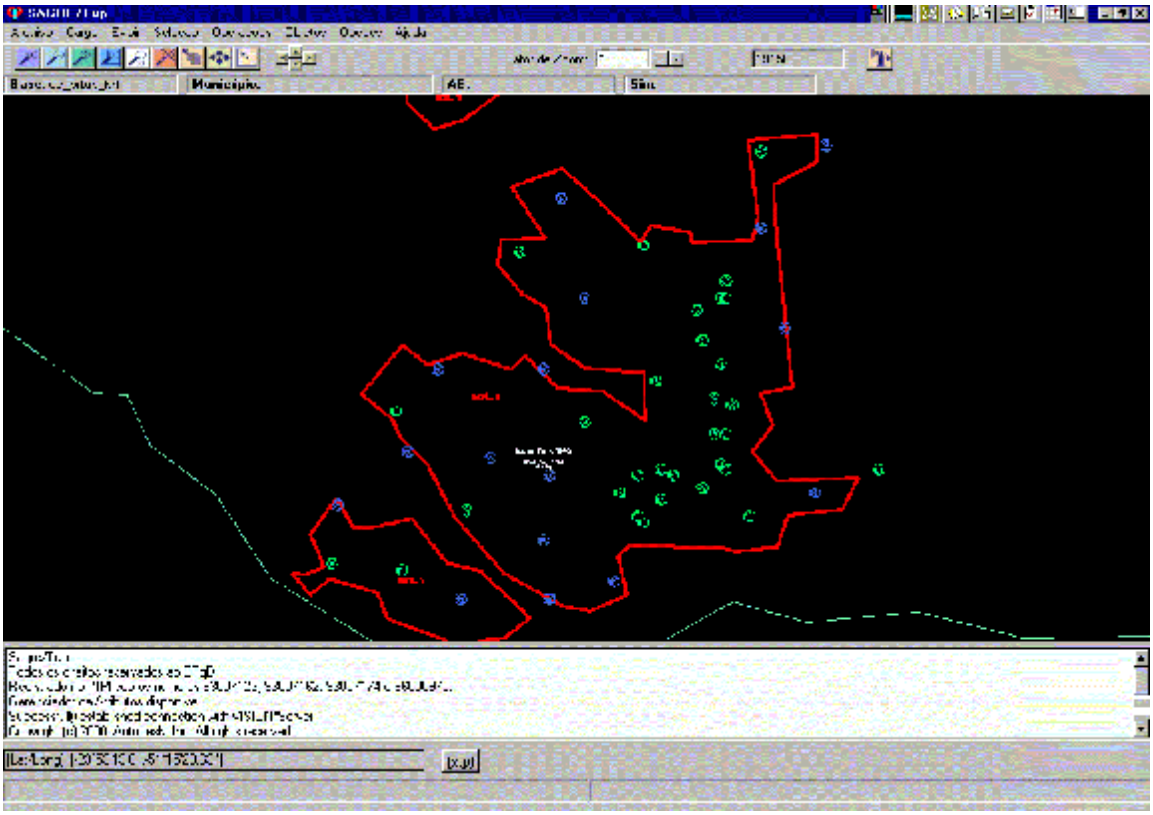


Figura 4.1: Representação de Telefones Públicos.

4.1.2 CPqD Supervisão Remota

O CPqD Supervisão Remota é o *software* responsável pela supervisão remota de um conjunto de terminais instalados de TPCI. Ele é um sistema capaz de supervisionar remotamente até 32000 telefones públicos a cartão indutivo por servidor, realizando a

monitoração constante das condições operacionais dos TPCIs contribuindo para a redução dos gastos com a manutenção da planta.

Os TPCIs são equipamentos eletrônicos baseados em microprocessadores que controlam o funcionamento dos aparelhos. As informações sobre o estado do aparelho podem ser obtidas através de um visor de cristal líquido, que fornece informações como o número de unidades de créditos restantes no cartão, o número que está sendo chamado, a necessidade de inserção ou troca do cartão e o estado operacional do aparelho. Outras informações como total dos créditos utilizados, entre outras, são enviadas automaticamente para uma central com o CPqD Supervisão Remota que aciona a execução das tarefas necessárias de acordo com os dados obtidos do aparelho.

Os TPCIs monitoram o seu funcionamento através de sensores internos, facilitando o controle das atividades de manutenção e instalação. Estes sensores detectam situações como: rompimento do cordão de monofone; abertura não autorizada do terminal e problemas com a leitora dos cartões. Além disso, eles também coletam dados sobre: bilhetes das chamadas realizadas; estatísticas de pequenas falhas; receita, utilização e identificação dos terminais. Estas informações são enviadas para o CPqD Supervisão Remota diariamente através de uma transmissão via modem numa taxa de 1200 bps. A Figura 4.2 mostra o relacionamento entre os componentes do CPqD Supervisão Remota.

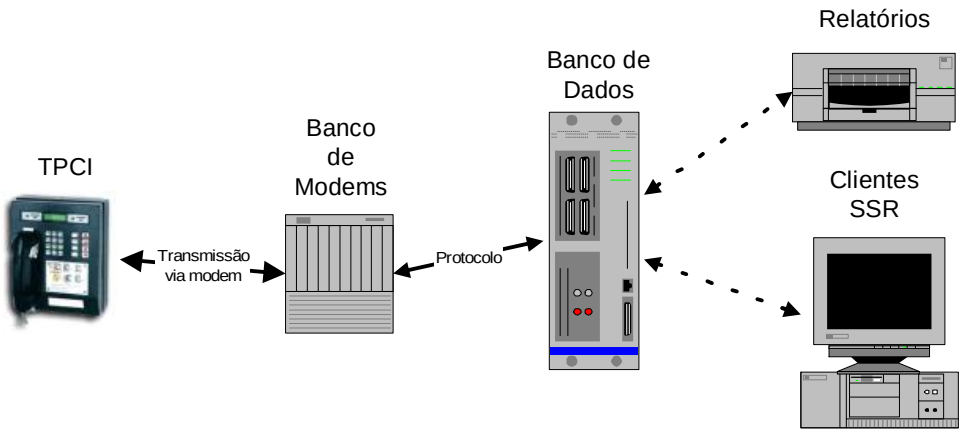


Figura 4.2: Relacionamento entre componentes do CPqD Supervisão Remota.

4.2 Arquitetura do Sistema

Atualmente existe um sistema de integração que funciona através de uma comunicação RMI onde o CPqD Supervisão Remota disponibiliza uma vez por mês um arquivo XML com os dados dos TPCIs, então os dados deste XML são convertidos para outro arquivo XML em um formato que o CPqD Gerência de Planta pode entender e estes dados são acrescentados a sua base de dados. A proposta para o protótipo é substituir esta comunicação RMI pela utilização de *Web Services*. O CPqD Supervisão Remota disponibilizaria serviços utilizando a tecnologia de *Web Services* para prover um acesso direto do CPqD Gerência de Planta a sua base de dados.

Hoje os dados dos TPCIs depois de inseridos na base do CPqD Gerência de Planta são tratados por um Gerenciador de Atributos (GAT) que pode exibir, entre outras informações, dados detalhados de um TPCI específico. Este gerenciador está escrito na linguagem de programação Java, ele faz acesso à base de dados do CPqD Gerência de Planta e disponibiliza estes dados para o usuário. Com a utilização dos *Web Services*, o gerenciador de atributos, ao invés de fazer consultas ao banco de dados do CPqD Gerência de Planta para recuperar alguns dados do TPCI, poderia fazer uma chamada direta ao *Web Service* que iria buscar estas informações no sistema CPqD Supervisão Remota e retornaria o resultado direto para o GAT, que iria exibir estes dados *online* de forma atualizada para o usuário.

Outra aplicação para os *Web Services* para a integração com o CPqD Supervisão Remota, seria através do módulo do CPqD Gerência de Planta chamado de *Web Viewer* que exibe em um *browser* os mapas com os objetos gráficos do CPqD Gerência de Planta. Uma nova funcionalidade poderia ser adicionada a este módulo para que ele possa exibir mapas temáticos dos TPCIs. Um mapa temático exibe objetos gráficos em cores e/ou formatos diferentes com base em uma faixa de valores predeterminados. Estes valores podem estar definidos em uma base de dados que possui informações que associa a identificação gráfica do objeto a uma das faixas de valores existentes. O *Web Viewer* poderia chamar um *Web*

Service para manipular os dados na base do CPqD Supervisão Remota de forma que eles possam ser disponibilizados para que o *Web Viewer* possa ler esta base de dados diretamente e exibir um mapa temático para o usuário.

O ganho do CPqD Gerência de Planta na utilização dos *Web Services* é que este faria acesso direto à base de dados do CPqD Supervisão Remota em tempo real, evitando a replicação destes dados na sua base de dados e tendo acesso as informações mais atualizadas. Este seria o melhor exemplo de interoperabilidade com a utilização dos *Web Services*, mas devido a alguns problemas administrativos, não será possível a implementação destes serviços com o acesso direto dos *Web Services* a base de dados do CPqD Supervisão Remota.

Portanto, para esta dissertação, realizamos uma simulação da base de dados do CPqD Supervisão Remota através de um arquivo XML e construímos os *Web Services* que oferecem serviços de consulta sobre os dados do TPCI e colocam estes dados a disposição do CPqD Gerência de Planta sem a necessidade de inseri-los em sua base de dados. Para a implementação real, foi construída uma página em HTML onde um usuário pode preencher um formulário simples, e esta faz uma consulta aos dados de um TPCI especificado por ele. O usuário entra com o número de serviço do TPCI, que é composto pelo número do prefixo mais o número do mcdu (milhar – centena – dezena – unidade), e o sistema retorna uma página com os dados solicitados.

Também foi incluída outra aplicação, mais complexa, com a implementação de mapas temáticos com a renda do TPCI, também baseada em dados XML. O usuário entra com o código da localidade desejada, então um serviço recupera estes dados e insere em uma base de dados do *Microsoft Access*, necessária para o *software* que exibe o mapa na Web, então uma página HTML retorna para o usuário um mapa temático com a localização dos TPCIs e uma classificação baseada na renda de seus créditos.

Foi utilizado o servidor Apache AXIS em conjunto com o *Apache Tomcat* para implementação dos *Web Services*, o IIS (*Internet Information Server*) como servidor de

Web e a linguagem Java com sua API para tratar os arquivos XML. A seguir apresentamos mais detalhes sobre as tecnologias utilizadas na implementação do protótipo.

4.2.1 Autodesk Mapguide

No presente trabalho utilizamos o *Autodesk MapGuide* para manipular as informações geográficas dos TPCIs e apresentar um mapa temático relativo aos dados de renda de cada TPCI, ou seja, é construído um mapa com as representações gráficas do TCPI de acordo com uma escala de valores definidos dentro da faixa de renda máxima e mínima de todos os aparelhos.

Este *software* é utilizado atualmente pelo módulo *Web Viewer* do CPqD Gerência de Planta e disponibiliza uma base de dados de um Sistema de Informação Geográfica (SIG) para usuários em uma Intranet/Internet. O SIG utiliza técnicas matemáticas e computacionais para o tratamento de informações geográficas e o *Autodesk MapGuide* possui funcionalidades que permitem realizar consultas e pesquisas nesta base. A maioria das soluções tecnológicas de publicação web para SIG na Internet usam imagens estáticas (por exemplo GIF ou JPEG - *Joint Photographers Expert Group*), já o *Autodesk MapGuide* utiliza dados cartográficos vetoriais baseados em objetos geográficos organizados em múltiplas camadas, permitindo ao usuário trabalhar como um cliente inteligente interagindo com os mapas e seus dados, controlando a visualização de camadas, impressão, seleção de objetos e recuperação de informações. Com a instalação de um *plug-in* no *browser*, o usuário pode utilizar ferramentas para consultar uma base de dados disponível na rede com suporte a consultas e atualizações com simplicidade e dinamismo por via remota.

Utilizamos três módulos do *Autodesk Mapguide* no presente trabalho:

- *Autodesk MapGuide Author*: Permite a criação e modificação dos dados espaciais nos mapas que são disponibilizados na Web. Ele é responsável pela confecção do mapa digital utilizado no projeto. Estes mapas são armazenados no formato MWF (*Map Window File*) e contém a completa especificação de visualização, incluindo as

características de cor de fundo, camadas dos dados cartográficos e especificações de configuração como menus e legendas.

- *Autodesk MapGuide Server*: Disponibiliza os dados dos mapas na Web, usando a interface padrão CGI (*Comon Gateway Interface*) e suportando um número ilimitado de requisições remotas. De acordo com o pedido do *plug-in* ele analisa as condições de segurança e envia o mapa via Web.

- *Autodesk MapGuide Viewer*: P*lug-in* que deve ser instalado no *browser* do usuário para permitir a visualização dos mapas. Ele interpreta os comandos enviados pelo usuário através de uma API disponibilizada pelo *software*, codificando estes comandos e passando para o *Autodesk Mapguide Server* que irá responder de acordo com o seu pedido [MAPGUIDE].

4.2.2 AXIS

Como servidor de *Web Services* utilizamos o AXIS apresentado no capítulo 2. A escolha deste servidor foi devido a sua característica de ser uma implementação modular, flexível e de bom desempenho do SOAP que atende perfeitamente os requisitos do nosso projeto. Como também ao fato dele pertencer ao projeto Apache especializado em fornecer soluções desenvolvidas usando o modelo de programação de código fonte aberto com padrões de qualidade comercial [AXIS].

4.2.3 Apache Tomcat

O *Apache Tomcat* é utilizado como um *servlet* container para a implementação da tecnologia de JSP (*Java Server Pages*). Ele é usado como um pequeno servidor *stand-alone* para executar as páginas JSP. A escolha deste servidor também foi devido ao fato dele pertencer ao projeto Apache e usar um modelo de programação de código fonte aberto [TOMCAT].

4.2.4 Internet Information Server

O IIS é uma ferramenta para criar uma plataforma de comunicação de aplicações dinâmicas na rede, gerenciando os serviços web na Internet. Ele é usado para hospedar e controlar páginas web, sites FTP e rotear *news* ou e-mails usando o NNTP (*Network News Transport Protocol*) e o SMTP. Neste projeto apresentado, o IIS é usado como servidor Web, onde ele recebe as requisições dos clientes e as direciona para o servidor que oferece o tratamento específico. Em nosso protótipo ele redireciona as requisições específicas dos mapas para o *Autodesk Mapguide Server*.

4.3 Diagramas do Protótipo de Integração

Neste tópico apresentamos os diagramas em UML (*Unified Modeling Language*) do protótipo desenvolvido para a integração dos *softwares*. O sistema é representado através de um conjunto de diagramas, onde cada diagrama se refere a uma visão parcial do mesmo, onde o conjunto forma um todo integrado e coerente.

4.3.1 Diagrama de Casos de Uso

O diagrama de casos de uso descreve a interação do cliente com as ações que o sistema que realiza. O usuário se encontra fora do escopo de atuação do sistema, enquanto o conjunto de casos de uso forma o escopo do mesmo. Ele representa graficamente esta interação, e define o contexto do sistema. Em nosso protótipo, o Cliente pode executar três ações no sistema: Consulta a Renda Diária, Consulta a Renda mensal e Mapa Temático de Renda Mensal (Figura 4.3).

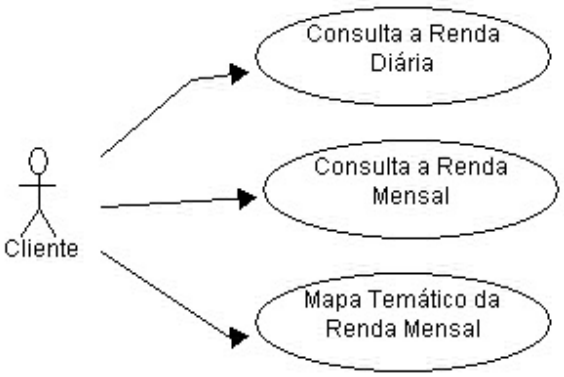


Figura 4.3: Diagrama de Casos de Uso.

4.3.2 Diagrama de Classes

O diagrama de classes descreve as classes que formam a estrutura do sistema e suas relações. Em nosso protótipo, as relações entre as classes *FormConsultaDia*, *FormConsultaMensal* e *FormConsultaTema* são de associações, indicando que as classes relacionadas interagem com a classe *Consultas* para solicitar seus serviços e obter os resultados que elas precisam para cumprir um objetivo para o sistema (Figura 4.4).

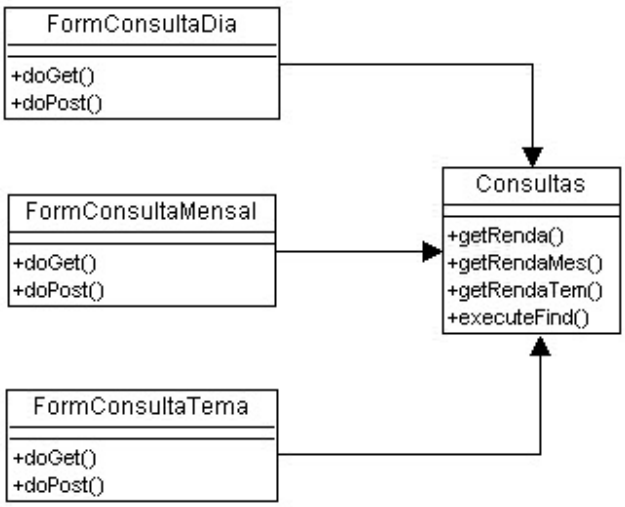


Figura 4.4: Diagrama de Classes.

4.3.3 Diagrama de Colaboração

O diagrama de colaboração mostra a colaboração dinâmica entre os objetos. Ele expõe as mensagens que são trocadas entre os objetos e apresenta os relacionamentos entre eles, descrevendo os cenários identificados pelos casos de uso, dando ênfase ao contexto do sistema. A seguir, apresentamos o diagrama de colaboração para o caso de uso do Mapa Temático da Renda Mensal (Figura 4.5). O diagrama exibe as mensagens de consulta TPCIs que expõe a solicitação do tipo de serviço para os objetos *servlets* e Web *Service*, exibe também a mensagem de Recupera TPCIs para a base XML, a mensagem InsereRenda TPCIs para a base *MS-Access*, a mensagem de RecuperaMapa para o servidor de mapas e a mensagem de RecuperaRenda TPCIs para a base *MS-Access* deixando claro qual tipo de operação deve ser feita em cada objeto e o tipo de resposta que eles devem fornecer, além de exibir a seqüência em que estas operações devem ser realizadas.

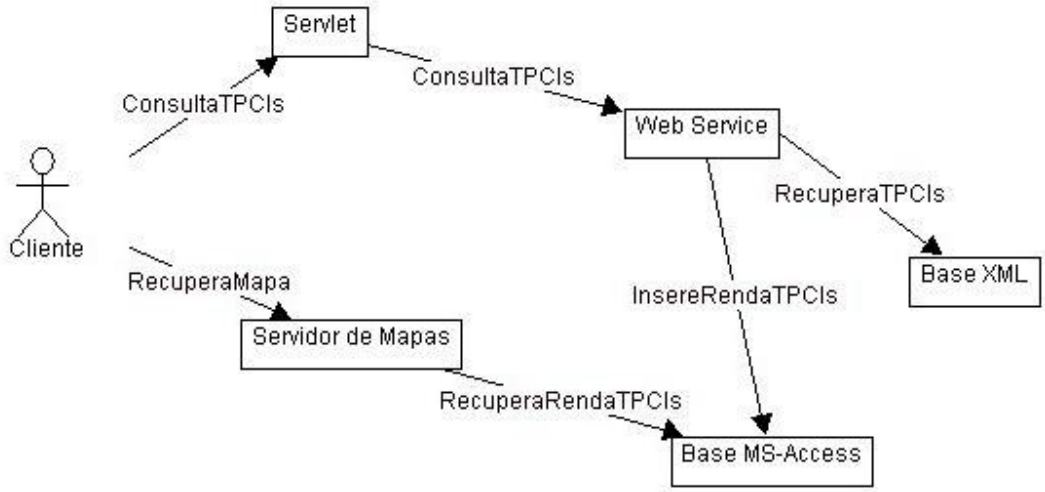


Figura 4.5: Diagrama de Colaboração.

4.3.4 Diagrama de Seqüência

O diagrama de seqüência mostra a colaboração dinâmica entre os vários objetos do sistema, apresentando a seqüência de ações executadas por ele e a seqüência de mensagens enviadas entre os objetos. No projeto desenvolvido o cliente entra com os dados dos TPCIs como o cnl (código nacional da localidade), o mês e o ano, que são enviados para um *servlet* no

servidor web. Após o processamento da requisição neste servidor, estes dados são enviados para o *Web Service*, que ao executar um serviço, consulta uma base XML para recuperar as informações necessárias a respeito do TPCI. Em seguida, após recuperar a renda do TPCI na base XML e processar esta informação, este serviço faz a inserção do resultado do processamento em uma base de dados *MS-Access*. Logo após, a *servlet* gera uma página HTML que é enviada como resposta para o cliente. Quando esta página chega ao *browser* cliente este faz uma solicitação ao servidor de mapas, que processa a requisição e envia o mapa resultante para o cliente. A Figura 4.6 apresenta o diagrama de seqüência para o caso de uso de Mapa Temático da Renda Mensal.

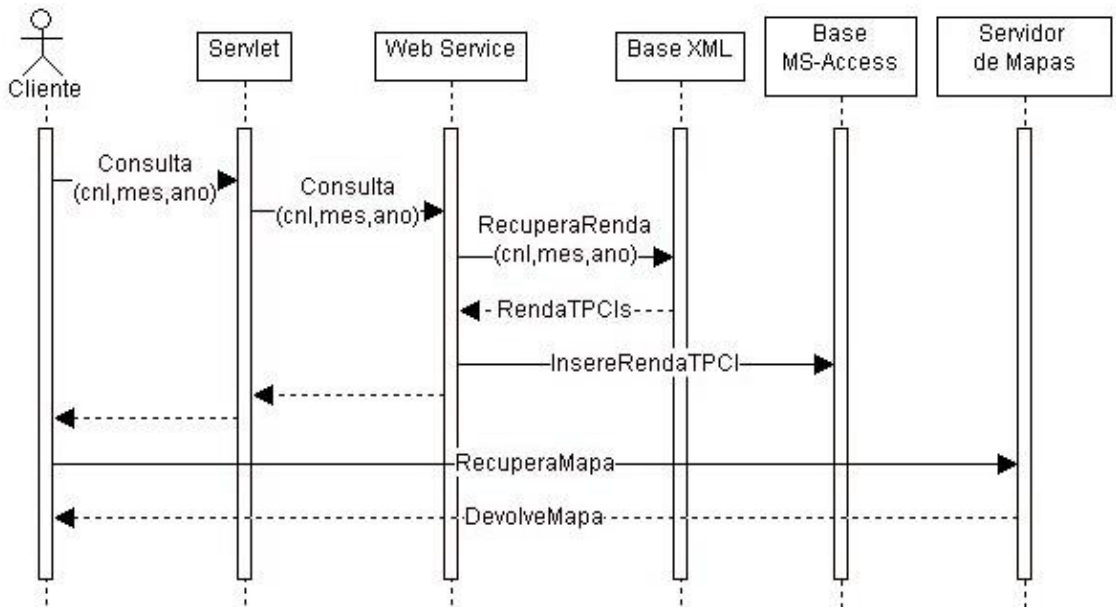


Figura 4.6: Diagrama de Seqüência.

4.4 Implementação

Nesta seção detalhamos o funcionamento da parte do sistema que apresenta o mapa temático sobre a renda. O usuário acessa a página HTML a seguir e escolhe um dos tipos de consultas que deseja realizar (Figura 4.7).

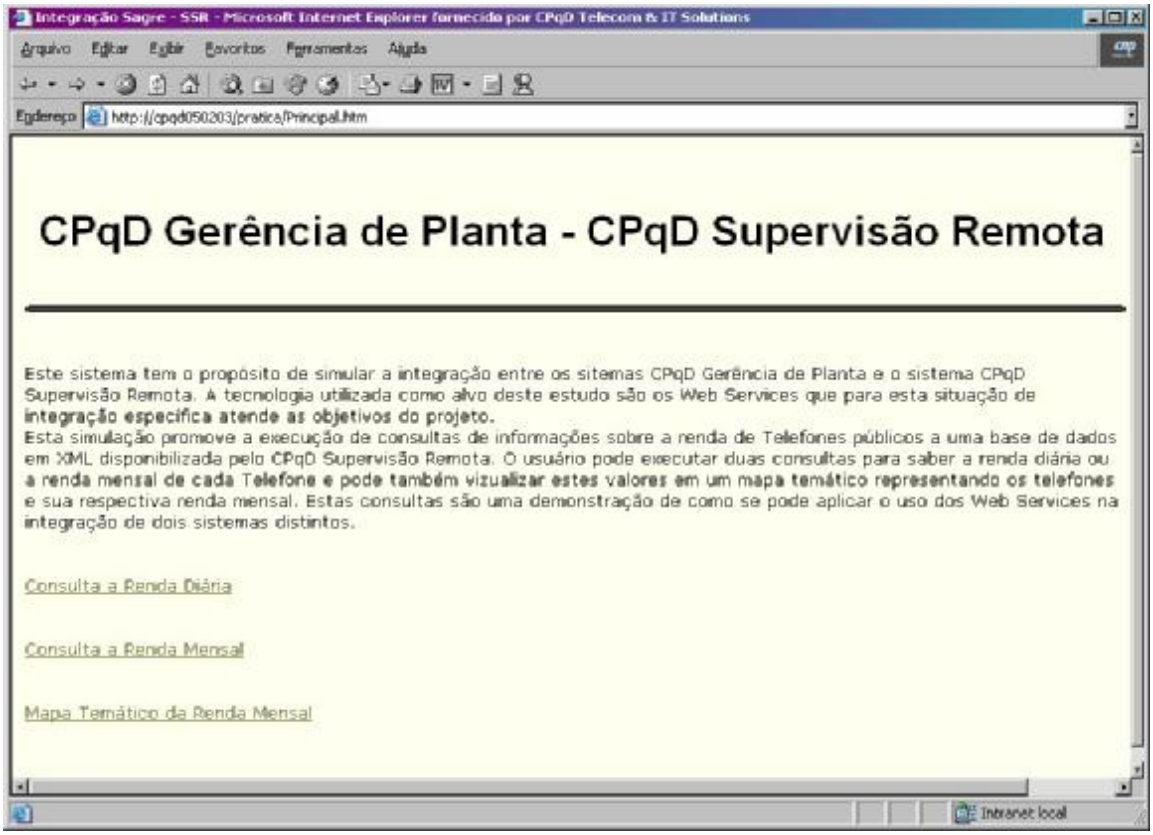


Figura 4.7: Tela Inicial do Protótipo.

Para o caso de consulta ao mapa temático o usuário deve escolher o *link* Mapa Temático da Renda Mensal, onde será exibido um formulário onde o usuário deve entrar com as informações do Código da Localidade, o Mês e o Ano sobre o qual deseja obter o mapa (Figura 4.8).

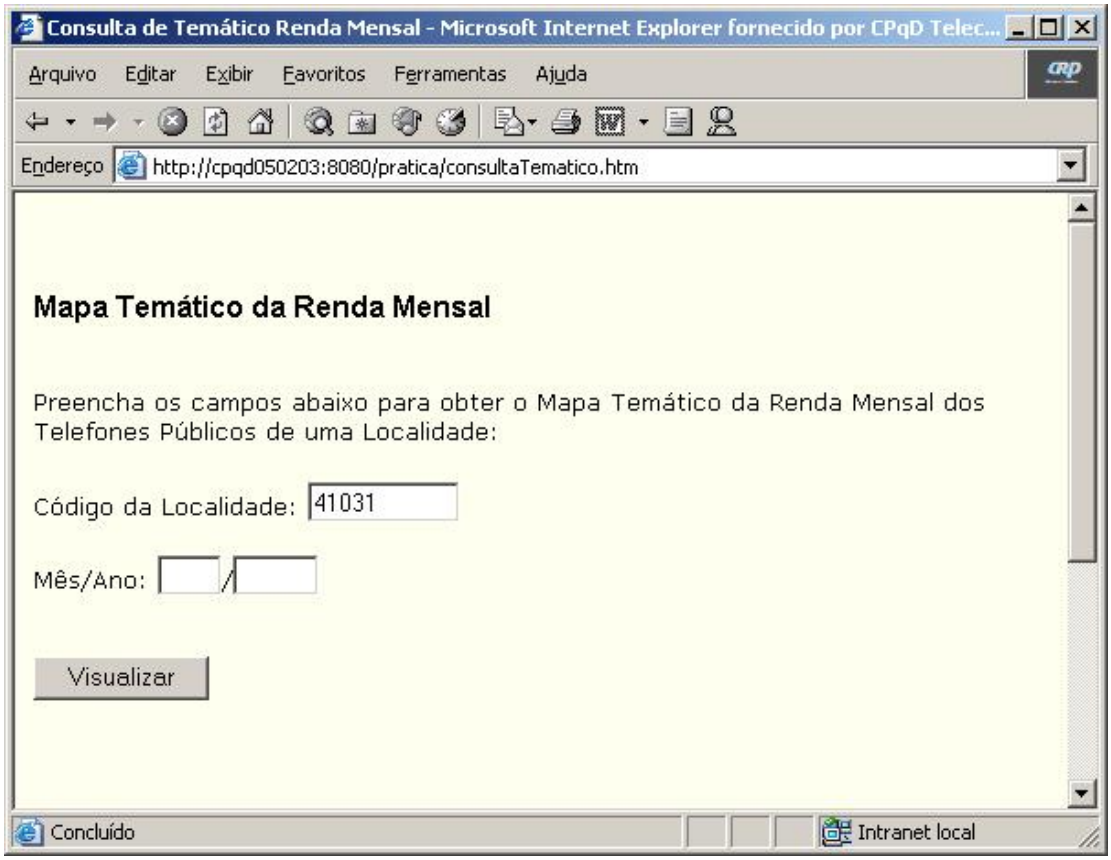


Figura 4.8: Tela de Entrada de Dados

Ao clicar no botão Visualizar, uma requisição é enviada para a *servlet FormConsultaTema.class* que solicita o serviço do *Web Service* e envia o resultado através de uma página HTML com o mapa temático solicitado. Os TPCIs são exibidos em diferentes cores, sendo que cada cor representa uma faixa de valores para a renda dos créditos a qual pertence o aparelho (Figura 4.9).

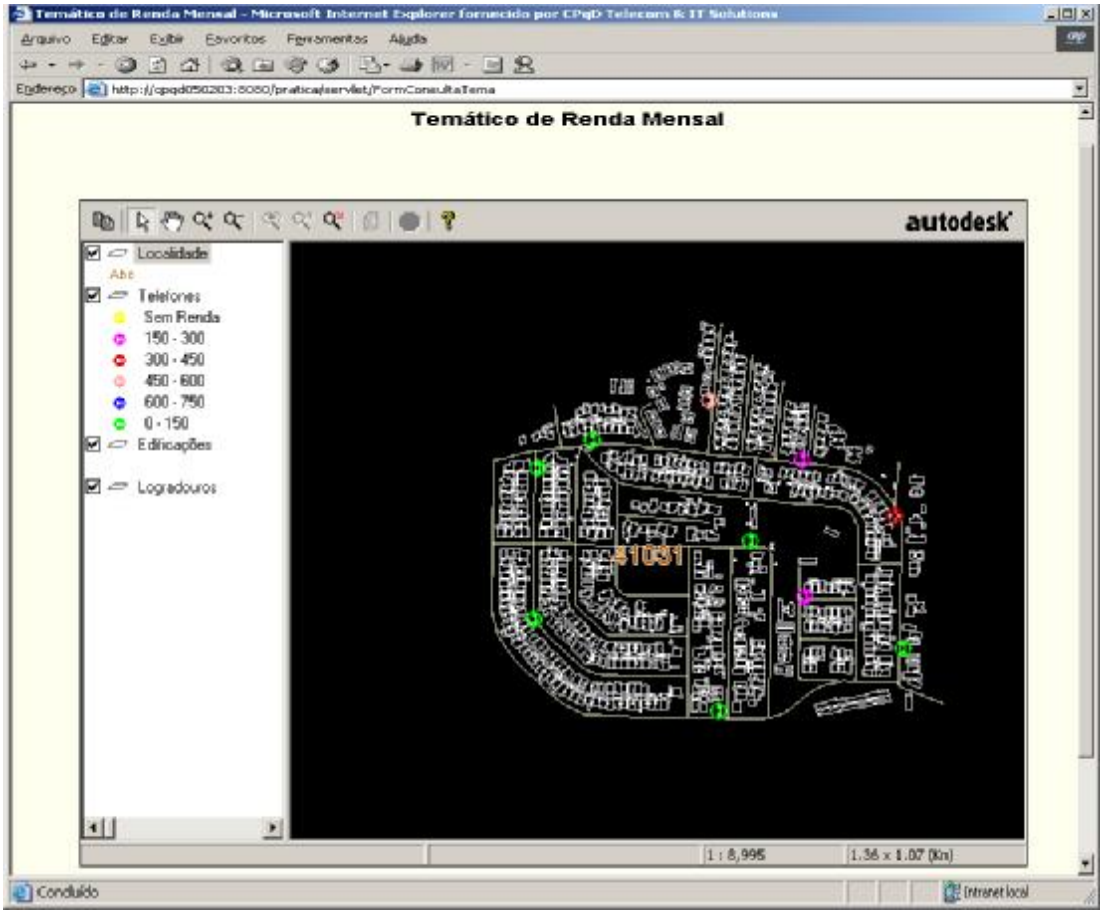
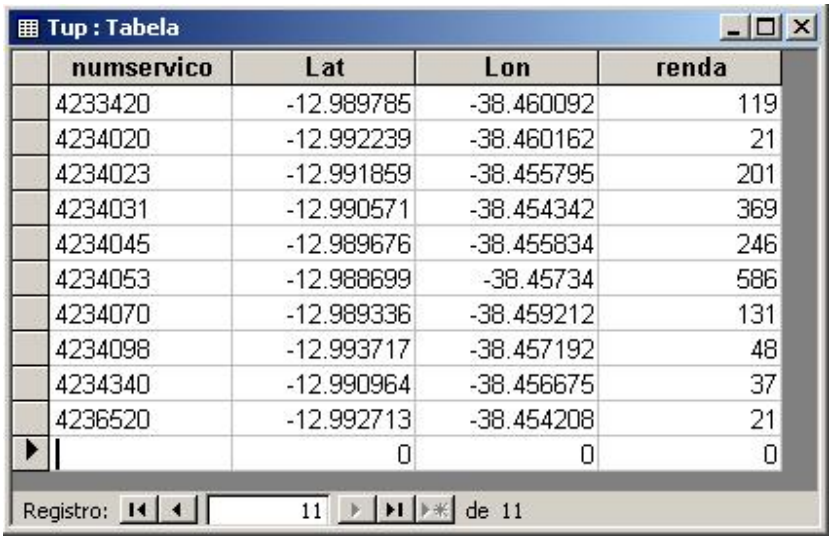


Figura 4.9: Tela com o Mapa Temático

A *servlet FormConsultaTema.class* faz a uma chamada para o *Web Service* passando como parâmetro o código da localidade, o mês e o ano. O serviço faz a consulta a base de dados em um arquivo XML, calcula a renda mensal de cada aparelho e envia o resultado para o banco de dados Tups.mdb. Ele preenche o campo renda na tabela Tup com o respectivo resultado para cada TPCI representado pelo seu número de serviço. Nesta tabela também se encontram as coordenadas geográficas de cada aparelho que são utilizadas pelo Autodesk Mapguide para exibi-los no mapa. A faixa de valores de renda e sua respectiva cor estão definidas no arquivo Mapa.mwf, cujo formato é específico para exibição no Autodesk Mapguide. Dependendo do valor da sua renda o aparelho será exibido em uma determinada cor no mapa. A Figura 4.10 mostra a tabela que foi preenchida pelo serviço e utilizada para construir o mapa temático.



Tup : Tabela

numservico	Lat	Lon	renda
4233420	-12.989785	-38.460092	119
4234020	-12.992239	-38.460162	21
4234023	-12.991859	-38.455795	201
4234031	-12.990571	-38.454342	369
4234045	-12.989676	-38.455834	246
4234053	-12.988699	-38.45734	586
4234070	-12.989336	-38.459212	131
4234098	-12.993717	-38.457192	48
4234340	-12.990964	-38.456675	37
4236520	-12.992713	-38.454208	21
	0	0	0

Registro: 11 de 11

Figura 4.10: Tabela de Informações dos TPCIs

Descrevemos a seguir parte do código cliente de chamada ao método *getRendaTem* no *Web Service* que é responsável pelo preenchimento da tabela Tup.

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;

String cnl = req.getParameter("cnl");
String mes = req.getParameter("mes");
String ano = req.getParameter("ano");

String endpoint = "http://"+req.getServerName()+":8080/axis/Consultas.jws";
String metodo = "getRendaTem";

try {
        Service  service = new Service();
        Call     call    = (Call) service.createCall();

        call.setTargetEndpointAddress( new java.net.URL(endpoint) );
        call.setOperationName(metodo);
        call.addParameter( "cnl", XMLType.XSD_STRING, ParameterMode.IN );
        call.addParameter( "mes", XMLType.XSD_STRING, ParameterMode.IN );
        call.addParameter( "ano", XMLType.XSD_STRING, ParameterMode.IN );
        call.setReturnType( XMLType.XSD_STRING );

       String renda = (String) call.invoke( new Object [] { cnl, mes, ano });
    }
```

Este código faz acesso ao *Web Service* definido no arquivo Consultas.jws. A classe Service deve ser usada como ponto inicial de acesso ao SOAP *Web Service*. Tipicamente, o serviço será criado com um documento WSDL e em conjunto com um *serviceName* pode-se então

solicitar uma chamada para o objeto *Call* permitindo a invocação do *Web Service*. Esta interface oferece suporte para a chamada dinâmica de uma porta de serviços. O objeto *org.apache.axis.client.Call* implementa a interface *javax.xml.rpc.Call*. Quando uma instância de *Call* é criada os métodos "set" definem a URL do endpoint da porta dos serviços de destino, o nome da operação e os nomes, os tipos e os modos (IN, INOUT ou OUT) dos parâmetros, e também o tipo do dado de retorno. Em seguida, usando o objeto *Call*, passando o cnl, o mês e o ano como parâmetros de entrada invocamos o serviço no *Web Service* que retorna a renda mensal.

O serviço definido no arquivo Consultas.jws é disponibilizado no servidor AXIS na URL http://cpqd050203:8080/axis/Consultas.jws, quando o usuário acessa o serviço acima, o AXIS automaticamente localiza o arquivo, compila a classe e converte as chamadas do SOAP para as chamadas Java da classe do serviço. Esta forma de distribuição do serviço não pode ser utilizada em qualquer situação, pois requer que o código fonte fique exposto no servidor. Para os casos onde temos apenas os arquivos de classe compilados utiliza-se um descritor de organização conhecido como WSDD (*Web Service Deployment Descriptor*) que descreve como os vários componentes instalados no AXIS se ligam internamente junto com as mensagens de entrada e saída do processo para o serviço.

O método *getRendaTem* definido no Consultas.jws acessa uma base de dados do CPqD Supervisão Remota simulada através do arquivo tups.xml que contém informações sobre a localidade que o TPCI pertence, o número de serviço do TPCI e informações sobre a data e a renda em créditos do aparelho. A seguir temos uma pequena amostra deste arquivo.

```
<?xml version="1.0" encoding='utf-8'?>
<!-- <!DOCTYPE PERFIL SYSTEM "Perfil_TUP.dtd"> -->
<!-- DADOS MENSAIS DE PERFIL DOS TUPS -->
<PERFIL>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4020"  dia="30"  mes="12"  ano="2002"
renda="10"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4020"  dia="29"  mes="12"  ano="2002"
renda="25"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4020"  dia="05"  mes="12"  ano="2003"
renda="8"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4020"  dia="26"  mes="12"  ano="2003"
renda="4"/>
```

```
    <TUP  cnl="41031"  prefixo="423"  mcdu="4020"  dia="29"  mes="01"  ano="2003"
renda="1"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4023"  dia="29"  mes="12"  ano="2002"
renda="23"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4023"  dia="09"  mes="12"  ano="2002"
renda="56"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4023"  dia="29"  mes="12"  ano="2002"
renda="45"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4098"  dia="09"  mes="12"  ano="2003"
renda="6"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4098"  dia="11"  mes="12"  ano="2003"
renda="3"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4098"  dia="15"  mes="12"  ano="2003"
renda="4"/>
    <TUP  cnl="41031"  prefixo="423"  mcdu="4098"  dia="29"  mes="01"  ano="2003"
renda="1"/>
</PERFIL>
```

Com base neste arquivo XML foram construídos os serviços que fazem as consultas às informações sobre os TPCIs neste arquivo e retornam estas informações para o cliente. Estes serviços foram desenvolvidos na linguagem Java e utilizam as APIs JAXP (*Java API for XML Parsing*) do Java para processar arquivos XML. Neste caso específico utilizamos a API DOM (*Document Object Model*) que fornece uma estrutura de árvore para os objetos. Esta API controla a hierarquia dos objetos na aplicação, sendo que ela é adequada para aplicações interativas porque o modelo inteiro dos objetos é carregado para a memória facilitando a manipulação destes. Ela foi utilizada nesta simulação devido a rápida resposta a pesquisa dos dados, já que eles estão todos na memória. Entretanto, se houvesse um grande volume de dados, a utilização desta API seria inadequada porque a aplicação poderia não suportar a carga completa dos dados na memória. Supondo que este modelo fosse aplicado realmente ao CPqD Supervisão Remota, o serviço deveria fazer uma consulta direta a base de dados dele sem precisar utilizar a API do XML para manipular os objetos de TPCIs, pois o volume de dados é muito grande. No caso de manter o uso do arquivo XML como base para consulta para grandes volumes de dados, então seria melhor utilizar a API SAX (*Simple API for XML*), que apesar de fazer uma pesquisa sequencial, ela não carrega todos os dados na memória e consequentemente a aplicação poderá suportar as consultas a estes dados. Apresentamos a seguir um trecho do código da função *executeFind* que faz a consulta a base XML:

```
org.w3c.dom.NodeList nodeList = root.getChildNodes();
for (int i=0; i<nodeList.getLength(); i++)
{
  org.w3c.dom.Node node = nodeList.item(i);
```

```
  int type = node.getNodeType();
  if (type == node.ELEMENT_NODE)
  {
       org.w3c.dom.NamedNodeMap nodeAtr = node.getAttributes();
       if (nodeAtr.getLength() > 0)
       {
         for (int j=0; j<nodeAtr.getLength(); j++)
         {
              org.w3c.dom.Node nodeA = nodeAtr.item(j);
              typeAtr = nodeA.getNodeType();
              nodeAName = nodeA.getNodeName();
              nodeAValue = nodeA.getNodeValue();
              if (typeAtr == nodeA.ATTRIBUTE_NODE)
              {
                if (nodeAName.equals("cnl") && nodeAValue.equals(cnl))
                {
                     isCnl = true;
                }
                if (nodeAName.equals("mcdu") && nodeAValue.equals(mcdu))
                {
                     isMcdu = true;
                }
                if (nodeAName.equals("prefixo") && nodeAValue.equals(prefixo))
                {
                     isPrefixo = true;
                }
                if (nodeAName.equals("mes") && nodeAValue.equals(mes))
                { isMes = true; }
                if (nodeAName.equals("ano") && nodeAValue.equals(ano))
                { isAno = true; }
                if (nodeAName.equals("renda")) { renda = nodeAValue; }
              }
          }//for
         if (isCnl && isMcdu && isPrefixo && isMes && isAno)
         {
              rendaAux = rendaAux + Float.parseFloat(renda);
         }
         isMcdu = false;
         isCnl = false;
         isPrefixo = false;
         isMes = false;
         isAno = false;
       }
  }//if Element
}//for
rendaMedia = String.valueOf(rendaAux);
return rendaMedia;
```

Esta função retorna a renda média de um determinado TPCI que foi passado como parâmetro. Ela vai varrendo a árvore do arquivo XML, identificando o número do TPCI, o mês e o ano solicitado pelo usuário e vai somando os valores da renda, retornando ao final a renda mensal dos créditos.

Para se obter o arquivo WSDL do serviço, basta acessar a URL http://cpqd050203:8080/axis/Consultas.jws?wsdl onde será exibido o código WSDL do

serviço de consultas. Veja a seguir alguns trechos deste arquivo referente a chamada do serviço que obtém a renda mensal e aciona os mapas temáticos:

```
<?xml version="1.0" encoding="UTF-8" ?>
-<wsdl:definitions     targetNamespace="http://cpqd050203:8080/axis/Consultas.jws"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://cpqd050203:8080/axis/Consultas.jws"
xmlns:intf="http://cpqd050203:8080/axis/Consultas.jws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

O elemento <definitions> contém os atributos que definem os namespaces usados no documento WSDL.

```
- <wsdl:message name="executeFindResponse">
  <wsdl:part name="executeFindReturn" type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="executeFindRequest">
  <wsdl:part name="root" type="xsd:anyType" />
  <wsdl:part name="nserv" type="xsd:string" />
  <wsdl:part name="cnl" type="xsd:string" />
  <wsdl:part name="mes" type="xsd:string" />
  <wsdl:part name="ano" type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="getRendaTemRequest">
  <wsdl:part name="cnl" type="xsd:string" />
  <wsdl:part name="mes" type="xsd:string" />
  <wsdl:part name="ano" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getRendaTemResponse" />
```

O elemento <message> contém a mensagem de chamada dos métodos com todos os seus parâmetros. Para isto, cada elemento <message> possui um ou mais elementos <part> que formam as partes reais da mensagem, contendo os parâmetros e se referindo a um dos tipos definidos no documento.

```
- <wsdl:portType name="Consultas">
- <wsdl:operation name="executeFind" parameterOrder="root nserv cnl mes ano">
  <wsdl:input message="impl:executeFindRequest" name="executeFindRequest" />
  <wsdl:output message="impl:executeFindResponse" name="executeFindResponse"/>
  </wsdl:operation>
- <wsdl:operation name="getRendaTem" parameterOrder="cnl mes ano">
  <wsdl:input message="impl:getRendaTemRequest" name="getRendaTemRequest" />
  <wsdl:output message="impl:getRendaTemResponse" name="getRendaTemResponse"/>
  </wsdl:operation>
  </wsdl:portType>
```

O elemento <portType> define a quantidade e agrupa conceitualmente as operações definidas no elemento <message>, e o elemento <operation> define estas operações construindo uma seqüência útil. No caso acima, as operações são definidas como uma solicitação de um serviço que resulta em um retorno de uma resposta, referenciando as mensagens de entrada e saída que a compõem.

```
- <wsdl:binding name="ConsultasSoapBinding" type="impl:Consultas">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"
/>
- <wsdl:operation name="executeFind">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="executeFindRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded" />
  </wsdl:input>
- <wsdl:output name="executeFindResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://cpqd050203:8080/axis/Consultas.jws" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getRendaTem">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getRendaTemRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded" />
  </wsdl:input>
- <wsdl:output name="getRendaTemResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://cpqd050203:8080/axis/Consultas.jws" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
```

O elemento <binding> descreve o mecanismo usado por um serviço para se comunicar com um cliente, especificando como um <porType> pode ser chamado. Este elemento repete as informações de um elemento <portType> e adiciona informações específcas ao protocolo SOAP em particular.

```
- <wsdl:service name="ConsultasService">
- <wsdl:port binding="impl:ConsultasSoapBinding" name="Consultas">
  <wsdlsoap:address location="http://cpqd050203:8080/axis/Consultas.jws" />
  </wsdl:port>
  </wsdl:service>
```

O elemento <service> contém as informações referentes ao servidor e os elementos <port> dentro do elemento <service> especificam para onde enviar as mensagens

descritas no documento, sendo que diversos elementos <port> podem ser definidos para cada protocolo descrito no elemento <binding>.

Em seguida temos o código de solicitação HTTP do SOAP gerado pelo servidor AXIS que foi transmitido pelo método POST, contendo no corpo o nome do método *getRendaTem* e os parâmetros com a definição de tipos e os valores necessários para invocar a execução deste serviço.

```
POST /axis/Consultas.jws HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1
Host: cpqd050203
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 488

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <getRendaTem soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <cnl xsi:type="xsd:string">41031</cnl>
   <mes xsi:type="xsd:string">12</mes>
   <ano xsi:type="xsd:string">2003</ano>
  </getRendaTem>
 </soapenv:Body>
</soapenv:Envelope>
```

A informação com a resposta do SOAP contém os códigos de estado do HTTP usados para comunicar o estado deste protocolo. Um código de estado 2xx indica que a solicitação do cliente, incluindo o componente do SOAP, foi recebida e compreendida com sucesso. O resultado do processamento do método *getRendaTem* vem especificado no corpo da mensagem através do *getRendaTemResponse*. Para o nosso caso o método executado no servidor não retorna resposta, mas se ocorresse um erro, o servidor HTTP deveria emitir uma resposta HTTP 500 "*Internal Server Error*" e a mensagem SOAP indicaria o erro através do método <Fault>. O exemplo abaixo mostra a mensagem SOAP de resposta do servidor.

```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=AACEDDE0259A2B7D4D18636F22D77328; Path=/axis
Content-Type: text/xml; charset=utf-8
```

```
Date: Tue, 07 Oct 2003 01:25:20 GMT
Server: Apache-Coyote/1.1
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <getRendaTemResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 </soapenv:Body>
</soapenv:Envelope>
```

4.5 Considerações Finais

Neste capítulo apresentamos uma solução para integração entre os sistemas CPqD Gerência de Planta e CPqD Supervisão Remota utilizando a tecnologia de *Web Services*. Diante deste contexto, esta tecnologia trará um ganho à integração porque a interoperabilidade está baseada na implementação de funções simples de consultas transportando pouco volume de dados, não existindo a necessidade de rigidez nas transações e de características de tolerância a falhas. Ela atende bem o propósito de comunicação devido à simplicidade dos requisitos do sistema, já que a aplicação não requer funcionalidades dos *Web Services* que ainda estão em desenvolvimento, como por exemplo, a manutenção do contexto da requisição no lado do servidor. Ela também facilita a disponibilização de serviços entre códigos desenvolvidos em linguagens diferentes, sendo um canal mais fácil e barato de distribuição e expondo a informação de forma padrão que poderá ser aproveitada por outros sistemas diferentes.

Entretanto, devemos considerar que se houver uma expansão na complexidade do sistema, os *Web Services* não poderão mais suprir todas as funcionalidades requeridas devido a algumas de suas limitações, e pode causar problemas de maior consumo de banda no trânsito das informações e aumento de complexidade. Portanto, deve-se observar exatamente em qual contexto esta tecnologia pode e deve ser utilizada, como no caso apresentado, onde ela pôde trazer maiores ganhos para a interoperabilidade das aplicações que solicitavam apenas poucos recursos de operação.

Capítulo 5

Conclusão

5.1 Considerações Finais

A tecnologia de *Web Services* vem sendo amplamente difundida como uma solução revolucionária para os problemas de integração entre os sistemas de computação. Com a proposta de oferecer serviços empacotados e distribuídos em uma rede onde podem ser acessados de qualquer ponto, eles despertaram a atenção dos grupos envolvidos com sistemas distribuídos. Devido às suas características e à falta de um padrão comum de comunicação, já que com a grande concorrência entre as empresas de tecnologia foram desenvolvidos diferentes protocolos de comunicação sem a preocupação de construir um padrão comum, os *Web Services* trouxeram grandes vantagens para a interoperabilidade dos vários sistemas. Através do seu acoplamento fraco, do uso do XML e do SOAP que pode ser transportado por intermédio de múltiplos protocolos, entre eles o HTTP, e da independência das linguagens de programação, a interação entre as implementações dos sistemas torna-se mais fácil. Diante disso, a necessidade de estudar os seus recursos e a forma como eles podem ser utilizados constitui um tópico de pesquisa importante.

Nesta dissertação, desenvolvemos um estudo comparativo da tecnologia dos *Web Services* com as principais tecnologias de sistemas distribuídos existentes no mercado, analisando como cada funcionalidade se apresenta em cada tecnologia. Apresentamos também, um estudo de caso onde foi construído um modelo de integração entre dois sistemas de telecomunicação e analisamos como a utilização dos *Web Services* puderam trazer benefícios para este caso.

Com base no estudo realizado, observamos que os *Web Services* não resolvem todos os problemas de integração, e apesar da grande difusão desta idéia no panorama dos sistemas distribuídos eles não são adequados para todas as situações. As empresas precisam entender

que eles são apenas uma ferramenta e não uma estratégia de negócio. Elas não devem alterar seus planos de negócios em função da utilização dos *Web Services* só porque eles estão em evidência, eles não substituem por completo as tecnologias que existem hoje e se ela não tiver uma boa estratégia eles não ajudarão muito.

Concluímos que os *Web Services* podem ser utilizados como objetos distribuídos, embora não se encaixem na definição clássica da orientação a objetos. Eles atendem a diversos requisitos de distribuição e realmente facilitam a comunicação entre os sistemas, porém muitos aspectos importantes de suas funcionalidades ainda deixam a desejar e a sua ampla utilização pode não suprir alguns requisitos fundamentais para os sistemas como segurança, transações e outros fatores. As outras tecnologias de objetos distribuídos ainda são necessárias e os *Web Services* devem funcionar como complemento para elas e não como competidores. A melhor solução é a utilização das tecnologias em conjunto, onde os *Web Services* atuem como objetos de serviços responsáveis pela comunicação entre os sistemas e deleguem as outras funções aos objetos de outras tecnologias para agirem nos pontos de maior complexidade onde eles ainda não podem atender.

5.2 Trabalhos Futuros

Ao longo do desenvolvimento desta dissertação foi possível observar que diversas linhas de pesquisas estão sendo desenvolvidas atuando no sentido de resolver as características insatisfatórias dos *Web Services* como transações, tolerância a falhas, segurança e outros. Um dos pontos encontrados que é muito interessante e pode dar continuidade a esta dissertação é o estudo da Orquestração e Coreografia de *Web Services*.

A orquestração e a coreografia de *Web Services* descrevem como eles interagem entre si em cada mensagem, incluindo a lógica do negócio e os pedidos de execução das interações entre os vários componentes do sistema. Cada *Web Service* solicita um serviço de outro *Web Service* definindo uma seqüência de mensagens que podem envolver múltiplos componentes dentro de um fluxo de negócio. A distinção entre orquestração e coreografia é que na orquestração existe um *Web Service* central que possui as informações de quais

serviços são necessários e coordena o fluxo de chamada para cada serviço. Na coreografia não existe um controle central, cada *Web Service* possui apenas a informação de qual serviço deve ser chamado após a sua execução para completar o fluxo de negócio, onde cada parte envolvida no processo de interação faz o seu próprio controle.

Na orquestração dos *Web Services* que executam por longos períodos de tempo deve existir o gerenciamento de exceções e da integridade das transações. A arquitetura deve levar em conta como os sistemas responderão se ocorrer um erro ou o serviço invocado não responder em um determinado tempo. No caso de acontecer algum erro, esta deve oferecer uma funcionalidade para gerenciar a integridade da transação. A orquestração dos *Web Services* deve ser dinâmica, flexível e facilmente adaptável para as mudanças que ocorrem nos processos de negócios. A reusabilidade é também um importante benefício dos *Web Services* necessária para compor um alto nível de serviços para os processos orquestrados.

Existem hoje três principais padrões para orquestração dos *Web Services*: BPEL4WS (*Bussiness Process Execution Language for Web Services*), WSCI (*Web Services Coreography Interface*) e BPML (*Business Process Management Language*). O BPEL4WS é uma especificação da IBM, Microsoft e BEA que modela o comportamento dos *Web Services* em um processo de interação de negócios. Para ele a arquitetura da orquestração coordena as várias atividades no processo e comunica aos sistemas quando ocorre um erro. O WSCI é uma especificação da Sun, SAP, BEA e Intalio que define uma linguagem de colaboração para os *Web Services* descrevendo as mensagens entre os componentes que participam do processo de colaboração. Esta suporta mensagens correlacionadas, regras de seqüências, manipulação de exceções, transações e colaboração dinâmica. O BPML é uma metalinguagem desenvolvida pela BPMI (*Business Process Management Initiative*) que descreve os processos de negócios. O BPML foi incorporado ao protocolo WSCI, onde o WSCI pode ser usado para descrever a interação pública e o BPML desenvolve as implementações privadas no contexto da coreografia dos *Web Services* [PELTZ2003].

Finalmente, além do estudo destes padrões, outros trabalhos podem ser desenvolvidos envolvendo tópicos como o modelo de suporte a conversação entre os *Web Services*

envolvendo a negociação entre as partes; a segurança no contexto da orquestração dos *Web Services* e o gerenciamento dos *Web Services* nos processos de negócios.

Referências Bibliográficas

[ARSANJANI2003] A. Arsanjani, B. Hailpern, J. Martin, P. Tarr, *Web Services: Promises and Compromises*, ACM Queue - Vol. 1, No. 1 - Março 2003.

[ATKINSON2002] B. Atkinson, G. Della-Libera, S. Hada, et al., *Web Service*s Security (WS-Security), Version 1.0 05, Abril 2002. http://www-106.ibm.com/developerworks/library/ws-secure/

[AXIS] Apache AXIS. http://ws.apache.org/axis/

[BTPP2002] Business Transaction Protocol Primer, An OASIS Committee Supporting Document, Version: 1.0, 3 Junho 2002.

[CIRNE2000] L. D. Cirne, R. Macedo, Uma Abordagem para Tolerância a Falhas em JAVA através de Comunicação em Grupo. *In: Anais do 18º Simpósio Brasileiro de Redes de Computadores (SBRC'2000)*. Belo Horizonte-MG. 16 páginas. Maio/2000.

[CORBA] Introduction to CORBA. http://developer.java.sun.com/developer/onlineTraining/corba/corba.html#co3

[COSTA2002] G. Costa, O Modelo de *Web Service*s — Como Desenvolver Aplicações em uma Nova Arquitetura de Software. *In:PBTR - Promon Tecnologia Business & Tecnology Review.* Ano 02 - nº 04

[DCOM1996] DCOM Technical Overview, Microsoft Corporation, Novembro de 1996. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp

[DIALANI2002] V. Dialani, S. Miles, L. Moreau, D. De Roure, M. Luck, Transparent Fault Tolerance for Web Services based Architectures. In Eighth International Europar Conference (EURO-PAR'02), volume 2400 of Lecture Notes in Computer Science, pp 889-898, Padeborn, Alemanha, Agosto 2002.

[FRMI] Fundamentals of RMI Short Course. http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html

[FATOOHI1997] R. Fatoohi, D. McNab, D. Tweten, Middleware for Building Distributed Applications Infrastructure (A State-of-the-art Report on

Middleware), NASA, *NAS Technical Report NAS-97*-026, Dezembro de 1997.

[GUNZER2002] H. Gunzer, Introduction to Web Services, Março de 2002. http://bdn.borland.com/java/webtech/0,1418,10018,00.html

[HAASE] K. Haase, Java Message Service Tutorial. http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/copyright.html

[HENDRICKS2002] M. Hendricks, B. Galbraith, et al., *Profissional Java Web Services*, Rio de Janeiro: Alta Books, 2002.

[HONDO2002] M. Hondo, N. Nagaratnam, A. Nadalin, Securing *Web Service*s. *In: IBM Systems Journal - New Developments in Web Services and E-commerce* - Vol. 41, No. 2, 2002.

[HOQUE1998] R. Hoque, *CORBA 3*, ed. IDG Books, 1998

[IETF] Internet Engineering Task Force. http://www.ietf.org/

[IBM] IBM WebSphere. http://www.ibm.com/br/products/software/websphere/

[JNDI] Java Naming and Directory Interface. http://java.sun.com/products/jndi/tutorial/TOC.html e http://java.sun.com/products/jndi/overview.html

[JNI] Java Native Interface. http://java.sun.com/products/jdk/1.2/docs/guide/jni/

[JRMI] Java RMI over IIOP. http://java.sun.com/j2se/1.4.1/docs/guide/rmi-iiop/

[JTS] Java Transaction Server. http://java.sun.com/products/jts/

[KREGER2001] H. Kreger, Web Services Conceptual Architecture (WSCA 1.0), IBM Software Group, Maio de 2001.

[LUNG2000] L. C. Lung, J. S. Fraga, R. Padilha, L. Souza, Adaptando as Especificações FT-CORBA para Redes de Larga Escala. *In: Anais do XIX Simpósio Brasileiro de Redes de Computadores - SBRC'2001 - SBC*. Belo Horizonte, MG - Maio de 2000.

[MAPGUIDE] Autodesk Mapguide. http://www.mapguide.com

[NET] Microsoft .NET. http://www.microsoft.com/brasil/dotnet/default.asp

[OMG] Object Management Group. http://www.omg.org/

[OMG2000] Object Management Group, “Fault-Tolerant CORBA Specification V1.0”, OMG document: ptc/2000-04-04, April, 2000.

[OTHMAN2001] O. Othman, C. O’Ryan, D. C. Schmidt, An Efficient Adaptive Load Balancing Service for CORBA, *IEEE Distributed Systems Online*, Vol. 2, Nº 3, Março 2001. http://dsonline.computer.org/

[PELTZ2003] C.Peltz, Web Services Orchestration, HP. Resource Dev Central. Hewlett Packard, Co. Janeiro 2003. http://devresource.hp.com/drc/topics/web_services.jsp

[ROQUE2000] V. Roque, J.L. Oliveira, CORBA, DCOM e JavaRMI - Uma Análise Comparativa, *EEI'99 (Encontro de Engenharia Informática 99 da O.E.)*, pp. 126-136, Universidade do Minho - Braga, Portugal, Dez, 1999. Publicado na Revista INGENIUM nº 46 de Março de 2000, pp. 87-91.

[SASAOKA2002] L. K. Sasaoka. Controle de Acesso em Banco de Dados Geográficos. Tese de Mestrado, Universidade Estadual de Campinas, Junho 2002.

[SUN] Sun ONE. http://br.sun.com/produtos-solucoes/software/amb_aberto.html

[TODD2002] S. Todd, F. Parr, M. H. Conner, A Primer for HTTPR - An overview of the reliable HTTP protocol, Julho 1, 2001, Updated Abril 1, 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-phtt/

[TOMCAT] The Apache Jakarta Project. http://jakarta.apache.org/tomcat/index.html

[TYAGI2003] S. Tyagi, M. Stevens, S. Mathew, et al., *Java Web Services Architecture*, Morgan Kaufmann Publishers, Abril 2003.

[WAGNER2001] B. Wagner, Web Services - Manage State in Web Services, *Visual Studio Magazine*. vol. 11, nº 13, ano 2001.

[W3C] World Wide Web Consortium. http://www.w3.org/

[W3S] W3Schools, SOAP Tutorial. http://www.w3schools.com/soap/default.asp

[WSI] *Web Service*s Interoperability Organization. http://www.ws-i.org/

[ZHONGWEI] L. Zhongwei, M. Xiangjiang, W. Yiwen, Extension of RMI with Group Communication. http://www.cs.cornell.edu/Info/Projects/JavaGroupsNew/studentprojects/rmi/report.html