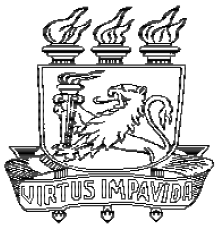

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ELETRÔNICA E SISTEMAS

MESTRADO EM ENGENHARIA ELÉTRICA

WEB SERVICES E J2EE: UMA SOLUÇÃO PARA INTEGRAÇÃO DE SISTEMAS EM AMBIENTES HETEROGÊNEOS

NÍVIA MENEZES DE OLIVEIRA

DISSERTAÇÃO DE MESTRADO

RECIFE - PE
27 de agosto de 2003

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ELETRÔNICA E SISTEMAS

NÍVIA MENEZES DE OLIVEIRA

WEB SERVICES E J2EE: UMA SOLUÇÃO PARA INTEGRAÇÃO DE SISTEMAS EM AMBIENTES HETEROGÊNEOS

Trabalho apresentado ao Centro de Tecnologia e Geociências da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica com Ênfase em Redes de Computadores sob a orientação do Prof. Rafael Dueire Lins, PhD.

Orientador: Prof. Rafael Dueire Lins, PhD.

RECIFE - PE
27 de agosto de 2003

NÍVIA MENEZES DE OLIVEIRA

WEB SERVICES E J2EE: UMA SOLUÇÃO PARA INTEGRAÇÃO DE SISTEMAS EM AMBIENTES HETEROGÊNEOS

Trabalho apresentado ao Centro de Tecnologia e Geociências da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica com Ênfase em Redes de Computadores sob a orientação do Prof. Rafael Dueire Lins, PhD.

Aprovado em 27 de Agosto de 2003.

BANCA EXAMINADORA

Rafael Dueire Lins, PhD
Orientador

Profa. Fernanda Maria Ribeiro de Alencar, PhD

Ricardo Massa, PhD

À minha amada família.

Agradecimentos

A Deus, por ter me mostrado o verdadeiro valor da vida durante o período que estive em Recife e por ter me ajudado a alcançar meu objetivo.

Aos meus pais, Valter e Dinamir de Oliveira, pelo grande incentivo e carinho demonstrados durante toda minha vida.

Aos meus irmãos, Silvana, Márcio, Marcelo, meus primos Zenila e Elton pelas força e apoio que me encorajaram a não desistir.

Ao meu eterno amigo e verdadeiro irmão, Álvaro Mota Gonçalves, que dedicou grande parte do seu tempo para tirar minhas dúvidas sobre Java e nunca se recusou em conceder sua ajuda, que foi de fundamental importância para elaboração deste trabalho.

Ao professor Rafael Dueire Lins, pois sem sua ajuda este trabalho não seria possível.

A todos meus familiares que sempre torceram por mim.

Em especial à Àurea Melo e Carla Oran que me ajudaram de muitas formas para a conclusão deste trabalho, principalmente com seus conselhos e compreensão, aceitando meus espinhos e fortalecendo ainda mais nossa amizade.

Aos integrantes da turma de mestrado, mais especificamente, Mirlem Ribeiro, Ennio Baptista, Jucimar Júnior, Manoel Azevedo, Raimundo Corrêa e Ricardo Barboza, que tornaram a estadia em Recife alvo de grandes recordações.

À Universidade Federal de Pernambuco, professores e funcionários (em especial ao Unilton Rodrigues), que direta ou indiretamente, ajudaram-me a atingir meu objetivo.

A todos que fizeram desta minha luta uma experiência de vida valiosa.

“Nunca mais direi eu não posso, pois tudo

posso Naquele que me fortalece.”

—

Resumo

A necessidade para resolução de seus problemas fez com que as empresas adquirissem sistemas nas mais diversas plataformas de desenvolvimento (linguagem de programação, SGBDs e Sistemas Operacionais), criando sistemas isolados que não possuem a capacidade de comunicação direta com os demais. Esse conjunto de sistemas chama-se ambiente heterogêneo. Nesses ambientes podem ser encontrados os sistemas legados que são sistemas antigos que desempenham um papel fundamental no cotidiano das empresas. As empresas que possuem um grande número de sistemas legados em ambiente heterogêneo precisam escolher uma estratégia para promover a comunicação entre tais sistemas. Esse desafio tem aumentado com o crescimento da Web, tornando-se cada vez mais importante e mais complexo, visto que a empresa necessita comunicar-se internamente, proporcionando maior intercâmbio de informações entre departamentos, e externamente, facilitando a interação com outras empresas. A busca por soluções para comunicação e troca de informações em ambiente heterogêneo originou a EAI (*Enterprise Application Integration*), que propõe uma série de tecnologias e processos para habilitar a comunicação entre os vários sistemas legados e a Internet. Necessidades cada vez maiores no sentido de tornar aplicativos de negócio interoperáveis têm conduzido o foco do desenvolvimento para o modelo da computação distribuída. Por isso, acredita-se que durante os próximos anos muitas empresas decidam integrar esses sistemas com novas aplicações em um ambiente distribuído, incorporando muitas das novas tecnologias disponíveis. O principal objetivo desta dissertação é estudar as caracteríticas de uma nova especificação para integração de sistemas chamada Web Services, comprovando sua capacidade de interoperabilidade e mostrando que é possível integrar diversos sistemas, independente da sua plataforma de desenvolvimento, utilizando tal especificação. Para tanto, serão estudadas as características dos Web Services e os recursos oferecidos pela plataforma J2EE para construção de Web Services que auxiliam no desenvolvimento dos mesmos. Também será implementada uma aplicação de integração de sistemas heterogêneos para verificação da interoperabilidade fornecida pelas tecnologias envolvidas.

Palavras-Chaves: Integração de Sistemas, Ambiente Heterogêneos, Interoperabilidade, Web Service, Java 2 Enterprise Edition.

Abstract

The necessity to solve problems made companies acquire systems in many development platforms (including programming language, DBMS and operating system), creating, this way, heterogeneous environments exchange information with others. Amongst those environments figure legacy systems, old system with a fundamental role in a company routine. Strategies to provide communication between systems are needed. This challenge has grown with the widespread use of Web, making this task more important and complex, because companies need to exchange information internally between departments, and externally, making easier the interaction with other companies. Due to system communication necessity and information exchange, EAI (*Enterprise Application Integration*) was born, it purpose is to provide a series of technologies and processes to enable communication between many legacy system and the Internet. Increasing necessities to make enterprise applications interoperable have conducted the development focus to distributed computing model. It is general belief that during next years, many companies will integrate these systems with new applications in a distributed environment. The main purpose of this dissertation is to study features of a new specification to system integration called Web Services, verifying Web Services interoperability and showing that it is possible to integrate many systems, independently of its original development platform, using Web Service specification. J2EE features that provide Web Services implementation are studied. An application to verify the interoperability provided by Web Services and J2EE platform is analyzed.

Keywords: System Integration, Heterogeneous Environment, Interoperability, Web Service, Java 2 Enterprise Edition.

Sumário

Lista de Figuras

Lista de Abreviaturas

API - Application Programming Interface
AWT - Abstract Window Toolkit
BI - Business Intelligence
BMP - Bean-Managed Persistence
B2B - Business to Business
COM - Component Object Model
CORBA - Common Object Request Broker Architecture
CRM - Customer Relationship Management
CMP - Container-Managed Persistence
CTS - Compatibility Test Suite
DCOM - Distributed Component Object Model
DD - Deployment Descriptor
DM - Data Mart
DOM - Document Object Model
DTD - Document Type Definition
DW - Data Warehouse
EAI - Enterprise Application Integration
EAR - Enterprise ARchive
EJB - Enterprise Java Beans
ERP - Enterprise Resource Planning
GUI - Graphic User Interface
HTML - Hypertext Markup Language
HTTP - HyperText Transfer Protocol
IDL - Interface Definition Language
JAR - Java ARchive
JFS - JavaServer Faces
JSP - JavaServer Pages
JVM - Java Virtual Machine
JAXB - Java Architecture for XML Binding
JAXM - Java API for XML Messaging
JAXP - Java API for XML Processing
JAXR - Java API for XML Registries
JAX-RPC - Java API for XML based Remote Procedure Call
J2EE - Java 2 Enterprise Edition
JMS - Java Message Service
JNDI - Java Naming and Directory Interface
JWSDP - Java Web Service Developer Pack
LDAP - Lightweight Directory Access Protocol
MOM - Middleware Orientado a Mensagem
ORBs - Object Request Broker
ORPC - Object Remote Procedure Call
PRM - Partner Relationship Management

RI - Reference Implemetation
RMI/IIOP - Remote Method Interface/ Internet Inter Orb Protocol
RPC - Remote Procedure Call
SAX - Simple API for XML Parsing
SCM - Supply Chain Management
SDK - Standard Development Kit
SGML - Standard Generalized Markup Language
SOAP - Simple Object Access Protocol
WAP - Wireless Application Protocol
WAR - Web ARchive
WML - Wireless Markup Language
WSDL - Web Service Description Language
W3C - World Wide Web Consortium
UDDI - Universal Description Discovery and Integration
UML - Unifying Modelling Language
URI - Unique Resource Identifier
XML - Extensible Markup Language
XSLT - XML Stylesheet Language Transformation

Capítulo 1

Introdução

Este capítulo descreve um breve panorama da origem dos problemas enfrentados pelas empresas que possuem sistemas em ambiente heterogêneo e desejam eliminar as ilhas de sistemas sem perder os serviços fornecidos por eles. Descreve também a motivação, os objetivos, bem como a idéia principal dos demais capítulos que compõem esta Dissertação.

1.1 Origem dos Ambientes Heterogêneos

O avanço do uso do computador nas diversas áreas de conhecimento, o número de sistemas de informação baseados em computador existentes nas organizações aumentou significativamente. As organizações adquiriram sistemas desenvolvidos nas plataformas mais adequadas às suas necessidades financeiras e técnicas.

Tais aplicações foram desenvolvidas por fornecedores diferentes, sem preocupação com a padronização, originando, assim, um ambiente heterogêneo. Ou seja, um ambiente que possui vários sistemas de informação desenvolvidos em diferentes linguagens de programação, SGBDs (Sistemas Gerenciadores de Banco de Dados), e sistemas operacionais. Percebeu-se, então, o problema de comunicação entre os sistemas, visto que as organizações necessitam manipular e integrar informações provenientes de todos os sistemas, independente da sua plataforma de desenvolvimento.

A existência de vários sistemas requer que a mudança realizada em uma informação comum seja feita manualmente em cada sistema relacionado com ela. Isso produz, eventualmente, dados redundantes e, muitas vezes, inconsistentes. A observação desses problemas, além do isolamento de dados foi uma das causas da necessidade de integração de sistemas. Além disso, as empresas precisam estender seu alcance, reduzir seus custos e tempo de resposta, fornecendo serviços de fácil acesso para seus clientes, parceiros, funcionários e fornecedores, bem como, aplicações de qualidade para criar novas oportunidades, integrando suas soluções de negócio.

Diante do cenário descrito, percebe-se que a integração entre sistemas em um ambiente heterogêneo é indispensável para o crescimento e desenvolvimento das organizações. Os benefícios de tal integração podem ser observados internamente, para acesso e compartilhamento de dados de sistemas legados entre os departamentos, e externamente, para comunicação com outras organizações.

A busca por soluções que viabilizassem a resolução do problema de integração deu origem à *Enterprise Application Integration* (EAI) [R+01b], que envolve uma série de tecnologias e processos para habilitar a comunicação entre os vários sistemas legados e a Internet. Como alternativa para possibilitar tal integração, pode-se citar os Web Services [A+02], pois os mesmos fornecem um padrão para comunicação entre diferentes aplicações, executadas em diversas plataformas e frameworks [CHA02].

Considerando que os Web Services requerem portabilidade, escalabilidade, segurança e eficiência, sua implementação deve ser feita utilizando-se uma tecnologia e uma plataforma que minimize o esforço de desenvolvimento e que forneça recursos requeridos. Para tanto, foi escolhida a plataforma J2EE [MIC02a], que se propõe a oferecer tais recursos por meio das APIs Java para XML [B+00], visando facilitar o desenvolvimento de Web Services.

Acreditando-se que o uso de Web Services pelas empresas vai permitir que sistemas de informação e as aplicações comerciais baseados em computador possam se comunicar de maneira a agilizar transações comerciais, será apresentada uma aplicação-cliente de Web Services na plataforma J2EE como uma alternativa para solucionar o problema referente à integração de ambientes heterogêneos.

1.2 Motivação

Devido aos problemas inerentes à existência de ambientes heterogêneos, é esperado que durante os próximos anos muitas empresas resolvam integrar esses sistemas com aplicações em um ambiente distribuído, incorporando muitas das novas tecnologias disponíveis.

Nesse contexto, interoperabilidade é a característica fundamental, haja vista que a preocupação em colocar os sistemas para interagirem entre si será um dos pontos mais fortes no desenvolvimento de sistemas nos próximos anos [BAN02]. Entende-se por interoperabilidade a habilidade de exprimir a transferência recíproca de usuários, dados ou informações, software e outros elementos de um sistema para outro [C+02].

Atualmente, a interoperabilidade está em um estágio pouco avançado. Poucos são os sistemas que conseguem interagir com outros para a realização de transações. Os sistemas que se enquadram nesse caso, conseguiram a muito custo, com soluções proprietárias que os deixou com pouca ou nenhuma flexibilidade, tais como CORBA [OMG03], MS.NET [LAU01] (COM+ e DCOM)[RAJ00] e a plataforma J2EE [R+01a], que inicialmente utilizava somente o EJB e RMI/IIOP [MIC02c], não oferecendo recursos para implementação de Web Services. Assim, a tecnologia de Web Services é uma nova alternativa para integração de sistemas, baseada em padrões da indústria de informática e com sua abordagem simplifica a colaboração em transações B2B (*business to business*). Dessa forma, acredita-se que o uso de Web Services vai permitir a comunicação entre sistemas de uma forma simples e, o que é melhor, seguindo padrões.

1.3 Objetivos

O objetivo desta dissertação é mostrar que a utilização da especificação de Web Services fornece uma solução interoperável para a integração entre sistemas.

Embora seja possível desenvolver Web Services em várias linguagens de programação ou plataformas de desenvolvimento, utilizou-se neste trabalho a plataforma J2EE por meio das APIs disponibilizadas no pacote Java Web Services Developer Pack (JWSDP) [ORT00] da Sun Microsystems. A plataforma J2EE foi escolhida para implementação dos Web Services em virtude de oferecer em sua infraestrutura recursos que proporcionam velocidade, escalabilidade, interoperabilidade, confiabilidade e segurança [COH01], facilitando o desenvolvimento das aplicações distribuídas.

1.3.1 Objetivos Específicos

Os objetivos específicos da dissertação são:

- apresentar os problemas de ambientes heterogêneos;
- apresentar os conceitos envolvidos na especificação de Web Services;
- estudar e utilizar os recursos da plataforma J2EE que auxiliam no desenvolvimento de Web Service;
- descrever um cenário de ambiente heterogêneo;
- implementar uma aplicação para integração de sistemas;

1.4 Organização da Dissertação

Nesta seção, será apresentada a organização e a visão geral de cada capítulo desta dissertação que é composta pelo capítulo introdutório, bem como pelos capítulos abaixo descritos.

No Capítulo 2 - Panorama Tecnológico dos Ambientes Heterogêneos, serão apresentados a contextualização do problema de integração de sistemas, as características referentes aos sistemas legados, bem como um breve histórico das aplicações, desde os ambientes centralizados até as tecnologias mais recentes, como Web Services. Posteriormente, serão descritos os conceitos envolvidos com a EAI (*Enterprise Application Integration*), a qual envolve uma série de tecnologias e processos para habilitar a comunicação entre os vários sistemas legados e a Internet.

O Capítulo 3 - Web Service apresenta os conceitos e as tecnologias envolvidas na especificação e desenvolvimento de Web Services, tais como XML [B$^+$00], SOAP [TEC02], WSDL [C$^+$03] e UDDI [Sec00].

No Capítulo 4 - Java 2 Enterprise Edition (J2EE), apresentam-se a descrição, a arquitetura e as características dos componentes que esta plataforma oferece, incluindo as APIs Java que possibilitam o desenvolvimento de Web Services.

No Capítulo 5 - Integração de Ambientes Heterogêneos - Um Estudo de Caso, será descrito um cenário, a modelagem por meio dos diagramas da UML [BEZ02], a descrição das ferramentas e estratégias de implementação utilizadas no desenvolvimento da aplicação de integração.

Já nas Considerações Finais, apresentam-se as contribuições e os resultados alcançados no trabalho, bem como as vantagens e desvantagem da solução. Também serão apresentadas algumas sugestões para futuros trabalhos na área de aplicações distribuídas e multi-camada para integração de sistemas.

Capítulo 2

Panorama Tecnológico dos Ambientes Heterogêneos

Este capítulo apresenta um panorama tecnológico dos ambientes heterogêneos. Primeiramente, serão apresentados a contextualização do problema de integração de sistemas, as características referentes aos sistemas legados por serem considerados como parte fundamental nos ambientes heterogêneos, bem como um breve histórico das aplicações, desde os ambientes centralizados. Posteriormente, serão descritos os conceitos envolvidos com a EAI (*Enterprise Application Integration*) cujo objetivo é fornecer uma série de tecnologias e processos para habilitar a comunicação entre os vários sistemas legados e a Internet. Por fim, são apresentadas algumas estratégias para integração de sistema, visando situar os Web Services em relação às demais soluções existentes.

2.1 Contextualização do Problema de Integração de Sistemas em Ambientes Heterogêneos

Durante vários anos as empresas compraram soluções em pacotes de software. Embora essas soluções funcionassem bem individualmente, eles criam verdadeiras ilhas de informações, produzindo, em muitos casos, informação redundante, além de requerer um trabalho manual que pode ser considerado ineficiente e cansativo.

A repetição e atualização do mesmo dado em vários sistemas, os resultados inconsistentes e os problemas de isolamento de dados foram os principais fatores para percepção da necessidade de troca de informação e integração de sistemas.

O cenário para necessidade de integração entre sistemas pode ser visto em várias situações.

Situação 1 - Fusão de empresas:

Assuma-se que duas empresas A e B desejem transformar-se em uma única empresa. Cada empresa possui suas próprias aplicações, sendo necessária a utilização dos dados e das funcionalidades de aplicações de ambas empresas. O que fazer para solucionar

esse problema? Existem várias soluções. A **primeira solução** seria fazer uma interface para cada sistema existente na empresa A com cada sistema existente na empresa B e vice-versa.

Sendo A1 e B1 aplicações, e m e n, o número de sistemas, das empresas A e B, respectivamente, e considerando que a interface entre A1 e B1 não é, necessariamente, igual à interface entre B1 e A1, pode-se concluir que o número de interfaces necessárias entre os sistemas das empresa A e B é $2*(m*n)$. Logo, o esforço empregado nessa solução é dispendioso. Além disso, as mudanças feitas nas aplicações podem afetar as interfaces, requerendo manutenção em vários programas diferentes.

A **segunda solução** seria abandonar as aplicações existentes e desenvolver uma única aplicação que unisse as funcionalidades de todas as aplicações existentes. A substituição dos seus sistemas legados por novos sistemas pode ser caro, lento e os novos sistemas podem não fornecer às empresas um apoio tão eficaz quanto o dos sistemas legados. Muitos sistemas legados ainda estão desenvolvidos na plataforma mainframe. Devido ao alto custo associado com a aquisição e manutenção de instalações do mainframe, o alto investimento nas aplicações e à alta confiabilidade do hardware mainframe, é difícil para as empresas justificarem a migração para uma arquitetura de servidor mais moderna.

Situação 2 - Aquisição de novos sistemas:

Considere-se uma empresa A que possui vários sistemas corporativos e deseja adquirir um novo sistema desenvolvido em linguagem de programação, sistema operacional e plataforma diferentes do nela já existentes. A solução para essa situação poderia ser a implementação de interfaces dedicadas para a integração dos sistemas legados com os novos sistemas. Como citado anteriormente, essa solução é dispendiosa e requer manutenção em diferentes interfaces em caso de mudanças nos sistemas envolvidos. Além disso, muitos sistemas legados ainda estão desenvolvidos na plataforma mainframe.

Como apresentado nas situações acima, além da necessidade de comunicação entre vários banco de dados, a necessidade de se ter várias aplicações interagindo tornou-se um problema para as empresas e para os desenvolvedores. Com milhões de linhas de código e o investimento correspondente em tempo de desenvolvimento e depuração das aplicações legadas, as organizações precisam de algo que reutilize as capacidades das aplicações existentes e faça com que os sistemas aprovados e consagrados pelo tempo de utilizaçào comuniquem-se de novas maneiras.

O desenvolvimento de aplicações distribuídas tem se tornado muito importante, ao mesmo tempo que a mudança do processamento de informações mudou dos computadores centralizados em mainframes para os minicomputadores e estações de trabalho em rede. Nesse contexto, surge a necessidade de comunicação e interoperabilidade entre aplicações.

O estudo de várias maneiras para realizar tal integração deu origem à Enterprise Application Integration (EAI), que será descrita com mais detalhes nas próximas seções.

2.1.1 Evolução de Aplicações Distribuídas

As aplicações na economia em rede costumam ser multicamadas, baseadas em servidor, permitindo interação com uma série de sistemas, e distribuídas, pois são executadas em vários dispositivos diferentes.

A ampla utilização de sistemas e dispositivos heterogêneos, bem como a extensão dos serviços oferecidos pelo servidor aumentaram a complexidade do projeto, desenvolvimento e distribuição das aplicações em rede.

As aplicações distribuídas mais antigas consistiam na utilização de mainframes, onde ficavam as aplicações, e em terminais para entrada e saída de dados. Essas aplicações eram centralizadas rodando em sistemas de computação com tempo compartilhado. Ou seja, um computador de grande porte (mainframe), contendo dados e aplicações de gerenciamento de dados, era conectado a uma série de terminais, cuja localização era baseada na abrangência permitida pela tecnologia. As raras redes usadas conectavam *mainframes* e eram lentas, mas esses sistemas eram de fácil desenvolvimento e manutenção, pois as aplicações eram homogêneas e todas residiam no mesmo ambiente.

Com o surgimento dos computadores pessoais PC (Personal Computer) houve a popularização das redes de computadores que pouco a pouco ganharam velocidade, viabilizando novas aplicações, inclusive causando a substituição dos mainframes numa política, nem sempre acertada, de downsizing. Nas redes de computadores os agentes são hierarquicamente iguais. O modelo cliente-servidor ganhou força e popularizou-se. Com isso, o poder de processamento para cada usuário foi aumentado, o processo de desenvolvimento de aplicação foi simplificado com várias ferramentas visuais e outros recursos de programação. Mas, a distribuição da aplicação nesse ambiente enfrentou problemas, pois o número de máquinas com configurações diferentes aumentou significativamente, tornando as redes heterogêneas trazendo novos desafios a serem equacionados.

Em seguida, surgiram as aplicações baseadas em navegadores na Internet ou em intranets, que são uma variação do modelo cliente-servidor. Um navegador executado em um PC de desktop possibilita o acesso ao servidor. As aplicações são executadas em servidores de aplicação, oferecendo toda a lógica comercial e manutenção de estado. Dessa forma, as aplicações podem oferecer desde simples pesquisa e navegação de página até processos dinâmicos mais complexos. O problema enfrentado nesse ambiente é referente à inclusão de funcionalidade, pois não existia um único padrão para clientes ou servidores, e as aplicações montadas desse modo são difíceis de se desenvolver ou manter.

Contudo, organizações tornaram-se mais exigentes e suas transações mais complexas, requerendo conexão entre diversos sistemas de *back-end*, por exemplo, conectar um sistema de estoque a um sistema de cobrança de cliente. Outro exemplo seria das empresas que se unem e precisam integrar seus recursos de computação herdados. Além de necessidade de comunicação entre vários bancos de dados, a necessidade de se ter várias aplicações interagindo logo tornou-se um problema, entre elas estão os sistemas legados, que serão descritos na próxima seção.

2.1.2 Sistemas Legados

Por observar o papel importante dos sistemas legados nos ambientes heterogêneos, faz-se necessário um estudo das suas características. Esta seção tem por objetivo apresentar tais características, bem como descrever os problemas trazidos por eles para a integração de sistemas.

Entende-se por sistemas legados os sistemas antigos desenvolvidos para uma empresa que resistem às modificações e evoluções no ambiente tecnológico, no entanto, não podem ser ignorados pelas organização em virtude dos dados e informações relevantes e de extrema importância [SOM95].

O dinamismo inerente ao cotidiano das organizações e ao ambiente tecnológico afeta diretamente os requisitos e funcionalidades oferecidos por um sistema de software. As razões para mudanças requeridas em um software podem ser classificadas em [S$^+$03]:

- evolutiva: o objetivo das mudanças é a melhoria do produto, adicionando novas funcionalidades solicitadas pelo usuário ou para aumento de desempenho e usabilidade;
- corretiva: as mudanças são feitas para consertar os erros no sistema;
- adaptativa: as mudanças são feitas com o intuito de acompanhar as mudanças nos ambientes, tais como sistemas operacionais, linguagens de programação, compiladores, sistemas de gerenciamento de banco de dados e outros componentes comerciais;
- preventiva: são as mudanças feitas para facilitar a manutenibilidade e melhorar a confiabilidade de um sistema.

As mudanças realizadas em sistemas pequenos que manipulam um pequeno número de atividades da organização não é preocupante, pois é possível re-projetar e substituir esse sistema sem comprometer as necessidades da organização. No entanto, os sistemas maiores requerem investimento para seu entendimento e sua manutenção e há uma preocupação em fazer com que os sistemas já existentes forneçam suporte aos novos sistemas [T$^+$03].

Entre os problemas enfrentados pelos desenvolvedores responsáveis pela manutenção de sistemas legados estão o tamanho, a complexidade e a falta de documentação dos sistemas. Além disso, muitas vezes, os sistemas legados utilizam linguagens de programação, sistemas gerenciadores de banco de dados e ferramentas de software obsoletas ou em desuso, sendo necessário treinamento de pessoal ou contração de profissionais especializados, demandando gastos adicionais para as organizações.

Um exemplo interessante referente à manutenção de sistemas legados trata da mudança de data em virtude da chegada de um novo milênio. As organizações contrataram muitos programadores e consultores especializados em COBOL, linguagem considerada até então em desuso, para fazer tal mudança em virtude do tamanho dos sistemas com necessidade de manutenção.

Um dos fatores para a rápida mudança no ambiente tecnológico pode ser associado ao surgimento de infraestrutura integradas, como a WWW, que possibilitou o acesso a sistemas anteriormente ilhados e de difícil acesso. Com o avanço da Internet e da Web, foi observada a necessidade de transportar dados entre as aplicações Web e os sistemas legados. Como exemplo, tem-se a necessidade de conectar os sistemas B2B com clientes de sistemas ERP (*Enterprise Resource Planning*) [PAC00].

Com a necessidade de buscar soluções e estratégias para o problema de integração entre sistemas, surgiu a EAI (*Enterprise Application Integration*), que será detalhada na próxima seção.

2.2 EAI - *Enterprise Application Integration*

EAI é o termo utilizado para descrever o planejamento, métodos e ferramentas necessárias para consolidar e coordenar diversos aplicativos de uma empresa. A EAI permite que aplicativos em plataformas heterogêneas (ex. ERP, CRM, Databases etc), muitas vezes localizados fisicamente em pontos distantes entre si, continuem sendo usados e gradativamente migrados e analisados dentro de uma nova visão de negócios e aplicações.

Essa tecnologia pretende potencializar uma nova conceituação e arquitetura de negócios, plataformas e aplicativos para uma empresa, inclusive os com padrão Web (Internet, Intranet, Extranet). Um fator crítico é observar como os sistemas e aplicativos legados se encaixam na nova visão concebida, analisando-se as possibilidades de sua utilização integral ou parcial, assim como o seu relacionamento com os novos aplicativos adicionados à nova plataforma tecnológica da empresa [LIN99].

A EAI é fundamental para a grande maioria das empresas que possuem sistemas e aplicativos heterogêneos e não integrados. Esta é uma realidade vivenciada principalmente por grandes corporações onde não só existe uma grande variedade de sistemas e

aplicativos mas também existem diferentes linguagens, plataformas e padronizações de comunicação entre si. Com a evolução dos negócios, da globalização e da competitividade cada vez mais acirrada, as empresas passaram a sentir a necessidade de integrar todos os sistemas e buscar soluções baseadas na Internet.

Nesse cenário de EAI, posicionam-se tecnologias genéricas como *drivers* de integração, *middleware* síncronos e assíncronos, gerenciadores de transações, *brokers* orientados a objetos, camadas de servidores Web, etc.

Dentre os motivos considerados relevantes para utilização da EAI estão [LIN99]:

- necessidade de integração dos sistemas ERP (*Enterprise Resource Planning*) com o restante dos sistemas sobreviventes da fase empacotada, e até mesmo integração entre os diferentes módulos ERPs;
- necessidade de integração dos sistemas CRM (*Customer Relationship Management*) e sistemas PRM (*Partner Relationship Management*).
- necessidade da definição de componentes integradores para prover interconexão do ambiente transacional dos pacotes e legados com o ambiente informacional dos *Data Warehouses* (DW) [INM97] e *Data Marts* (DM) [INM97], devido ao crescimento da área de BI (Business Intelligence);
- necessidade de cruzamento de fronteiras entre as aplicações de negócio na Internet, uma abordagem mais rápida e integrada entre as peças de sistemas já existentes e as novas que estão chegando. O SCM (*Supply Chain Management*) é um exemplo, pelo fato de representar os processos de produção e entrega do produto final, do fornecedor para o cliente;
- a adição de novas tecnologias indutoras de integração, como XML (*Extensible Markup Language*) e LDAP (*Lightweight Directory Access Protocol*), pode servir como motivação para a melhoria do ambiente de negócios eletrônicos da empresa. XML é uma proposta para intercâmbio de documentos, baseado em uma definição de metadados (tags auto definidos), que será descrita com mais detalhes no Capítulo 3 - Web Services. A outra abordagem, LDAP, é uma proposta para padronização de acessos aos diversos serviços de diretórios diferentes espalhados pelos ambientes de rede e de desenvolvimento, essa abordagem consiste em objeto de estudo desse trabalho.

O conceito de EAI pretende tornar possível a comunicação entre CRM, SCM, ERP, BI, DW, sistemas legados e outras propostas vindouras. Para tanto, são propostos alguns modelo de integração, tais como o Modelo de Integração de Dados, o Modelo Integração de Apresentação e o Modelo de Integração Funcional, descritos na próxima seção.

2.2.1 Modelos de Integração

A principal meta do EAI é permitir que uma organização integre diversas aplicações de forma rápida e fácil, além de reduzir o acoplamento. Segundo [R+01b], a integração pode ocorrer em três pontos de uma aplicação: nas camadas de apresentação, funcional ou de dados.

2.2.1.1 Modelo de Integração de Dados

Tem o objetivo de extrair os dados de uma base, podendo eventualmente modificá-los e fazer a atualização numa outra base [PAC00].

Um modelo de integração de dados permite a integração por meio do acesso aos dados que são criados, administrados, e armazenados tipicamente pelo software, com a finalidade de reuso ou sincronização dos dados através de aplicações [R+01b].

Diversas ferramentas e *middleware* [LIN01], aplicativo que interliga outros dois aplicativos que normalmente encontram-se em camadas diferentes, são usados para esconder as complexidades do sistema operacional e da rede que estão por baixo a fim de facilitar a integração de vários sistemas. No entanto, o acesso aos dados tem sido usado apenas para acessar e integrar informações de um banco de dados.

Como exemplos dessas ferramentas e de *middleware* temos [LIN99, LIN01]:

- *tranferência de arquivo em lote*- ferramentas que permitem mudar arquivos entre sistemas e aplicações de maneira *ad hoc* ou pré-determinada, sendo considerada como uma das primeiras ferramentas utilizadas para a integração de dados.
- *Open Database Connectivity* (ODBC, padrão da Microsoft amplamente utilizado para conectividade de banco de dados) - uma aplicação padrão de programação de interface que acessa bancos de dados relacional heterogêneos. Provê uma interface que pode ser usada para integrar banco de dados que fornece suporte a sua utilização.
- *middleware de acesso a banco de dados* - um tipo de software que facilita o acesso ao banco de dados através do uso ou criação de conectores. Além disso, disponibiliza o ambiente em tempo de execução para gerenciar as requisições enviadas a esses bancos de dados e retorno dos resultados. Forma de prover conectividade a banco de dados distribuído. *Middleware* de acesso de dados é focalizada na troca de consultas, gerenciamento de resultados, conectividade para banco de dados, agrupamento de conexões e outras tarefas de housekeeping (administração interna de negócios) relacionados a gerenciamento de dados.
- *transformação de dados* - uma ferramenta que usualmente complementa o *middleware*. Provê a habilidade de converter dados do formato do banco de dados fonte

no formato do banco de dados alvo. Devido ao fato de definições de dados, estruturas e esquemas serem comumente diferentes para cada uma das aplicações, a troca geralmente exige conversão.

Cada uma dessas tecnologias pode ser usada para solucionar problemas pontuais ou combinada a fim de resolver problemas de integração de dados mais complexos.

O Modelo de Integração de Dados deve ser empregado quando houver a necessidade de combinar dados de múltiplas fontes para análise e tomada de decisão, prover múltiplas aplicações com acesso à leitura a uma fonte de informações comum, bem como permitir que dados sejam extraídos de uma fonte e reformatados e atualizados em outra.

Esse modelo de integração de dados promove integração rápida quando os bancos de dados são facilmente acessíveis através de interfaces ou quando o *middleware* provê integração de dados de múltiplas fontes para aplicações novas.

Esse modelo também permite que dados sejam reutilizados através de outras aplicações. Cada integração é ligada a um modelo de dados. Se o modelo de dados mudar, a integração pode se desfazer. Nota-se, portanto, que a integração de dados é sensível a mudanças. Uma vez que sistemas tendem a evoluir, estas mudanças podem requerer um esforço significativo para se manter a integração.

2.2.1.2 Modelo de Integração de Apresentação

Utiliza as interfaces disponíveis nas aplicações para acessar processos ou simples informações [PAC00]. Este modelo de integração é baseado no conceito de acesso às aplicações de sistemas legados através de sua lógica de interface existente. Permite desenvolver uma interface de usuário nova através do mapeamento das antigas interfaces [R$^+$01b].

Considerado como o método de EAI mais primitivo. Os processos de negócio são unidos através de acesso ao programa. Esta forma de EAI pode ser gerenciada facilmente e a lógica existente é reutilizada. Nesse modelo, a integração de componentes de software múltiplos é realizada por meio de uma interface de aplicação do usuário. A nova apresentação simula ser uma única apresentação embora possa estar acessando muitas aplicações. Na integração lógica, tem-se as instruções para onde direcionar as interações do usuário, comunicar a interação do usuário ao software apropriado usando apresentações existentes como ponto de integração.

Esse modelo deve ser aplicado quando houver necessidade de:

- apresentar uma interface em que o usuário percebe como uma única aplicação porém, sendo de fato, a composição de várias aplicações;

- colocar uma interface de usuário baseada no PC em uma aplicação baseada em terminal existente, para prover uma aplicação de fácil manuseio para um usuário final; e
- integrar com uma aplicação cujo único ponto de integração útil e implementável é através de sua apresentação.

A integração de apresentação é muito fácil de ser realizada e pode ser feita de forma relativamente rápida. A lógica de apresentação é normalmente menos complexa que a lógica de dados ou funcional porque pode ser vista, é descoberta facilmente, e normalmente é bem documentada. Se o usuário dispuser de boa ferramenta para executar este trabalho de integração, a maior parte do trabalho necessário será executado pela ferramenta. Cabe ao usuário então, enfocar sua atenção na construção da nova apresentação. Por outro lado, a integração de apresentação ocorre somente no nível da interface do usuário. Conseqüentemente, somente dados e interações definidas nas apresentações legadas podem ser acessadas. E ainda, a integração de apresentação pode ter seu desempenho comprometido devido a camada extra de software para a aplicação existente. Os dados e a lógica subjacente da aplicação existente não podem ser acessados. Este tipo de integração é o mais limitado dos três. A integração ocorre na apresentação e não na interconexão entre as aplicações e os dados.

2.2.1.3 Modelo de Integração de Funcional

Usa os mecanismos que permitem o compartilhamento de métodos entre aplicações, como por exemplo, os objetos distribuídos [PAC00].

Esse método de integração requer que o ponto de integração seja no código da aplicação. Este ponto de integração pode ser tão simples quanto o acesso por meio de uma API (*Application Programming Interface*, conjunto ou biblioteca de rotinas e/ou funções que podem ser utilizadas por outro aplicativo) ou tão difícil como requerer código adicional para criar pontos de acesso [PAC00].

Já as chamadas de procedimentos remotos requerem que o ponto de integração seja no código da aplicação, entretanto, faz uso de um *middleware* de processamento distribuído, um tipo de software que facilita a comunicação de solicitações entre componentes de software através do uso de interfaces definidas ou mensagens, e provê um ambiente em tempo de execução para gerenciar os pedidos entre os componentes de software. Chamadas de procedimentos remotas proverão somente as definições para acesso e capacitação de comunicações básicas.

Tipicamente requerem esforço significativo de uso no desenvolvimento de softwares. *Middleware* provê uma abordagem mais robusta que combina definição de interface e de comunicações como também suporte em tempo de execução para administrar os pedidos

entre componentes de interface. Existem três categorias de *middleware* de processamento distribuído [LIN99]:

- *Middleware* Orientado a Mensagem (MOM) - provê integração por meio da passagem de mensagens entre as aplicações. Similar ao conceito de correio, uma mensagem é colocada no MOM. O MOM é então responsável pela entrega da mensagem ao sistema alvo. MOMs podem ser implementados em várias configurações inclusive fila e passagem de mensagens.

- *Tecnologia de Objeto Distribuído* - aplica o conceito de orientação a objeto ao *middleware*. Interfaces fazem com que o software assemelhe-se a objetos. O software pode então ser acessado por outras aplicações através da rede por meio das interfaces objeto.

- *Monitores de Processamento de Transação* - provê suporte às missões críticas para arquiteturas distribuídas por meio da permissão de uma transação ser gerenciada usando conceitos como *commit* de duas fases. Preservam a integridade dos recursos de informações distribuídas como banco de dados, arquivos e mensagens.

O modelo de integração funcional é mais flexível do que o de dados ou de apresentação. Pode ser largamente aplicada usando três diferentes abordagens. Cada abordagem apresenta característica diversa e são usadas para solucionar um tipo diferente de problema de integração. As abordagens são [LIN99, LIN01]:

- consistência de dados - integração através do código de uma aplicação onde a finalidade é acessar ou atualizar dados. A integração facilita a comunicação de dados e ações. Tipicamente é implementada enviando-se uma solicitação a cada sistema que descreve a ação que é pretendida junto com os dados. Este pedido pode ser passado para outras aplicações quando necessário.

- processo multi-passos - uma ação é processada corretamente por todas as aplicações pertinentes na ordem correta de precedência sem intervenção humana ou redigitação da informação. Integração de aplicações onde não há somente comunicações de solicitações como também a coordenação e gerenciamento dessas solicitações através das aplicações. A integração facilita a comunicação da solicitação e gerencia o fluxo e o sequenciamento.

- componente *plug-and-play* - conceito de que software pode ser criado como componentes com interfaces fáceis de se entender que podem ser conectadas facilmente para formar uma nova aplicação. Criação de interfaces reusáveis através de aplicações que simplificam a construção de novas aplicações. Este tipo de integração é o mais complicado das diferentes abordagens, porém tem o maior valor para o empreendimento.

O modelo de integração funcional provê a mais robusta integração entre todos os modelos. Pode ser usado para solucionar problemas dos modelos de dados e apresentação. Provê o reuso de componentes criados mais do que os outros modelos. Entretanto, este modelo é mais complexo.

No contexto dos modelos de integração definidos pela EAI, o objeto de estudo deste trabalho, o Web Service, pode ser classificado como uma estratégia de integração funcional, por ser uma tecnologia de objetos distribuído, cuja abordagem é caracterizada como componentes *plug-and-play*.

É importante salientar que existem outras tecnologias de objetos distribuídos disponíveis no mercado, tais como CORBA [OMG03], DCOM [RAJ00] e RMI [MIC99], cujas características serão descritas na próxima seção.

2.3 Tecnologias de Objetos Distribuídos

A programação em sistemas distribuídos é inerentemente complexa. Todos os benefícios relacionados devem ser alcançados com a implementação de um ambiente de programação bastante sofisticado e preciso. Para a construção de tais sistemas, a programação orientada a objetos oferece vantagens bastante desejáveis, como modularidade e extensibilidade [L+96].

Nos últimos anos diversos padrões foram propostos para a implementação do conceito de sistemas distribuídos. Um modelo muito popular tem sido o de separação de uma aplicação em camadas, tais como apresentação, negócio e dados. Esse modelo, bastante usado na Internet, tem por base uma infinidade de protocolos, dos quais DCOM [Soc01], CORBA [OMG03] e RMI [MIC02c] são os mais conhecidos. Embora estes protocolos sejam muito diferentes entre si, seus problemas são basicamente os mesmos: complexidade de implementação e, principalmente, implantação. Sistemas escritos em diferentes linguagens, e não raro em plataformas distintas, sofrem ainda de um mal bem mais grave, a incompatibilidade entre os tipos de dados, além da necessidade de forte envolvimento das equipes técnicas de ambos os lados no processo de integração. Essas têm sido as principais causas da baixa integração entre os sistemas de clientes e fornecedores mesmo com toda a facilidade de infra-estrutura de comunicação existente atualmente.

A seguir serão descritas resumidamente as características do DCOM, CORBA e RMI.

2.3.1 CORBA

A tecnologia CORBA (*Common Object Request Broker Architecture*) é uma solução aberta de objetos distribuídos desenvolvida pela OMG (*Object Management Group*) para

se tornar um padrão no mercado[OMG03].

A primeira vantagem do CORBA é que a implementação do cliente e do servidor pode ser feita em qualquer linguagem. Isso só é possível por causa de um alto nível de abstração conseguido através de um arquivo chamado IDL (*Interface Definition Language*). Esse arquivo possui o mapeamento dos métodos implementados no servidor, em uma linguagem específica. A comunicação entre objetos, cliente e servidores é feita por meio dos ORBs (*Object Request Broker*), responsáveis pela tradução das soluções e respostas. O IDL, por sua vez, se responsabiliza pela tradução das solicitações e respostas, mas possui uma grande limitação, pois suporta apenas os tipos de dados comuns a todas as aplicações.

Se o aspecto de desempenho for considerado muito importante, o CORBA é uma das soluções mais recomendadas. No entanto, para as aplicações pequenas, a complexidade de implementação o torna não muito atrativo [RAJ00].

2.3.2 DCOM

O DCOM (*Distributed Component Object Model*) [Z+00] é uma tecnologia desenvolvida pela Microsoft para distribuir objetos pela rede. O DCOM faz uso de uma tecnologia chamada COM (*Component Object Model*). Um servidor DCOM publica seus métodos para clientes através de múltiplas interfaces, escritas em IDL semelhante ao C++. O compilador IDL cria os *stubs* e os *skeletons* similar ao compilador CORBA, mas seu registro fica implantado no registro do próprio sistema operacional.

O protocolo utilizado para comunicação entre objetos para essa tecnologia é o ORPC (*Object Remote Procedure Call*). Esse protocolo faz uso do *ping* para permanecer com os objetos ativos, sabendo da atividade do cliente em tempo de execução. Suporta também o coletor de objetos distribuídos, conhecido como *garbage collection* [L+96].

Algumas empresas implementaram o DCOM junto com a Microsoft, possibilitando a portabilidade entre elas, como a Apple, Unix e VMS [MUE01, RAJ00].

2.3.3 RMI

O RMI (*Remote Method Invocation*) [MIC99] é uma facilidade que permite a programas escritos em Java chamar certos métodos em um servidor remoto. A interface RMI permite que objetos Java em hosts diferentes comuniquem-se entre si. Cada objeto remoto implementa uma interface remota que especifica quais de seus métodos podem ser invocados pelos clientes. Os clientes podem invocar métodos de um objeto remoto quase exatamente da mesma maneira que eles invocam métodos locais [D+01].

As aplicações RMI são geralmente compostas por dois programas separados: um servidor e um cliente. Uma aplicação-servidor típica cria um número de objetos remotos, faz as referências aos objetos remotos acessíveis e espera os clientes invocarem os métodos de objetos remotos. Uma aplicação-cliente típica pega uma referência remota para um ou mais objetos remotos no servidor e então invoca os métodos através dessas referências. O RMI fornece o mecanismo pelo qual o servidor e o cliente se comunicam e trocam informação entre si. Por isso, as aplicações RMI são chamadas aplicações de objetos distribuídos [MIC02c].

Para o desenvolvimento de clientes remotos, o uso de IDL não se faz necessário, tendo em vista que o próprio servidor é usado para difundir os objetos implementados. Isso faz com que a implementação de aplicações se torne cada vez mais simples, em relação ao CORBA. Por outro lado, apenas a linguagem Java é suportada em ambos os lados da comunicação, limitando bastante a amplitude de necessidades a serem alcançadas [RAJ00].

2.3.4 Comparação entre as tecnologias

De acordo com as características apresentadas anterior, pode-se observar que todas as tecnologias oferecem soluções similares. A diferença entre elas está em seus recursos, bem como em sua complexidade.

Como cada tecnologia opera de forma diferente, uma aplicação DCOM não pode se comunicar com uma aplicação RMI e assim por diante. Além disso, é recomendada para essas tecnologias a abrangência de uma intranet, e em alguns casos, o uso de *firewalls* para encriptar suas solicitações pela Internet, reduzindo seu desempenho e a consistência da comunicação [RAJ00].

A proposta dos Web Services para integração entre sistemas distintos é bastante simples e direta, baseando-se no protocolo SOAP [COH01]. Basicamente um Web Service funciona como uma página Web, com a diferença que ao invés de HTML, usa-se XML. Desta forma os dados podem ser descritos e o pacote da mensagem pode ser manipulado com grande facilidade tanto por quem envia, quanto por quem recebe [SUO02, KAO02].

Para um maior entendimento sobre a especificação de Web Service, no próximo capítulo são apresentados todos os aspectos e tecnologias envolvidos.

Capítulo 3

Web Services

Em termos gerais, Web Services são aplicações que oferecem serviços via Web. Visando um entendimento mais amplo sobre este tópico, neste capítulo serão apresentados a definição, a arquitetura e os recursos envolvidos na utilização de Web Services, tais como as XML, WEB SERVICEDL, UDDI e o protocolo SOAP que possibilita o seu desenvolvimento.

3.1 Introdução

A utilização de Web Services [A+02] está se expandindo rapidamente devido à crescente necessidade de comunicação e interoperabilidade entre aplicações.

Segundo [W+02], existem várias interpretações para o conceito de Web Services. Para alguns, um Web Service é simplesmente uma aplicação Web que executa um serviço. Para outros, um Web Service deve lidar com transações financeiras. E ainda existem os que acham que um Web Service é uma aplicação que usa o protocolo SOAP [BHA00]. Mas, nenhuma das definições citadas retrata o que um Web Service é realmente.

De acordo com a definição da W3C (*World Wide Web Consortium*) [A+02], um Web Service é um sistema de software identificado por uma URI (*Unique Resource Identifier*), cujas interfaces públicas e ligações são definidas e descritas usando XML. Suas definições podem ser descobertas por outros sistemas. Esses sistemas podem interagir com o Web Service da maneira descrita na sua definição, usando mensagens baseadas em XML transportadas através de protocolos da Internet [A+02].

Os Web Services consistem em uma especificação de padrões que pode ser implementada por diversos fabricantes. Os Web Services são caracterizados segundo os seguintes aspectos:

- descrição - os Web Services descrevem suas funcionalidades e atributos, para que outras aplicações possam descobrir como usá-los;
- publicação - os Web Services são registrados em um repositório que contém informações sobre como conectar e usar tais Web Services;
- descoberta - quando um Web Service é localizado, uma aplicação remota pode invocar o serviço;

- retorno de resposta - quando um Web Service é invocado, os resultados são retornados para a aplicação solicitante.

Os detalhes sobre a arquitetura dos Web Services serão descritas na seção seguinte.

3.1.1 Arquitetura de Web Services

Os Web Services fornecem um padrão para comunicação entre diferentes aplicações, executadas em diversas plataformas e frameworks [CHA02]. A arquitetura definida para um Web Service foi projetada para promover interoperabilidade e extensibilidade entre as várias aplicações, plataforma e frameworks de uma maneira que mantém-se consistente com a arquitetura Web.

A arquitetura de referência para Web Service identifica os componentes funcionais, define a relação entre eles e estabelece um conjunto de regras sobre cada um para atingir as propriedades desejadas as arquitetura completa [CHA02].

Segundo [COY00], a arquitetura de Web Service abrange três aspectos principais:

- um fornecedor de serviço que fornece uma interface para o software que realiza um conjunto de tarefas;
- um solicitante do serviço que descobre e invoca o serviço do software para fornecer uma solução para o seu negócio;
- um repositório que gerencia e publica o serviço. O fornecedor do serviço publica seus serviços no repositório e o solicitante acessa tais serviços criando ligações com o fornecedor.

A arquitetura de Web Service é apresentada na Figura 3.1. Primeiramente, o Fornecedor de Web Service usa o UDDI para registrar o Web Service em um repositório. Os Web Services disponíveis nesses repositórios são descritos em uma linguagem chamada WSDL. Os clientes, por sua vez, usam o UDDI para encontrar o Web Service adequado. A forma como o cliente se comunica como o Web Service depende do WSDL. Todas as comunicações são feitas por meio de arquivos XML trafegados sobre o protocolo SOAP.

A XML (*Extensible Markup Language*) é uma meta-linguagem que fornece um conjunto de regras e roteiro para descrição de dados estruturados [COY00].

o SOAP (*Simple Object Access Protocol*) é um protocolo baseado em XML que permite a comunicação entre o solicitante e o fornecedor [TEC02].

o WSDL (*Web Service Description Language*) descreve os serviços disponibilizados através da utilização do XML. Descreve a funcionalidade e como acessar um web service [COY00].

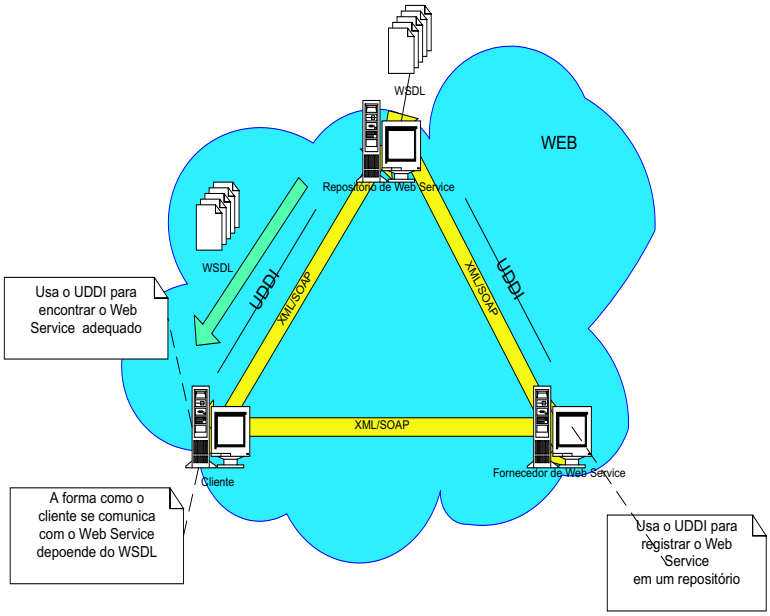

Figura 3.1. *Arquitetura de Web Services*

o UDDI (*Universal Description, Discovery and Integration*) é um padrão desenvolvido para prover informações em forma de diretório sobre os web services. Representa um repositório de serviço que habilita as requisições de serviço para que seja encontrado um fornecedor do serviço [COY00].

A seguir serão descritas tecnologias envolvidas, tais como XML, SOAP, WSDL e UDDI com maiores detalhes com o intuito de facilitar o entendimento sobre os Web Services.

3.2 XML - Extensible Markup Language

XML [B+00] é uma linguagem de marcação que descreve os dados através de etiquetas definidas pelo usuário, sendo os dados descritos através de DTDs (Documento de Definição de Tipo) ou do Esquema XML. Por ser baseada em texto, independente de qualquer ferramenta de hardware e software, a XML está se tornando o padrão para descrição e transmissão de informações. Documento é o termo usado para uma coleção de dados XML. Pode ser armazenado como um arquivo, um objeto ou em um banco de dados.

Muito mais que um conjunto de etiquetas e regras simples, a XML proporciona a evolução da computação comercial e como ela afeta muitos aspectos do desenvolvimento de software e comércio eletrônico.

3.2.1 Características e Objetivos

A XML é uma meta-linguagem definida pelo W3C que consiste em um conjunto de regras e guias para descrever dados estruturados em formato texto e não utiliza representações binárias proprietárias.

Por ser uma meta-linguagem, ela possui recursos para criar outras linguagens baseadas na inserção de etiquetas para descrever dados.

Assim como no HTML (*Hypertext Markup Language*), cada etiqueta deve ter um início e um fim. No exemplo apresentado a seguir, tem-se a etiqueta EXEMPLO como a primeira etiqueta na hierarquia, chamada de elemento raiz, após a mesma, tem-se a etiqueta Estudante, que, por sua vez, possui as seguintes etiquetas Nome, Tel, Fax e E-Mail. Cada etiqueta tem significado para os humanos e cada uma vem seguida do conteúdo válido para os mesmos. Por isso, diz-se que a XML é a combinação das etiquetas e os conteúdos, na qual as etiquetas dão significado ao conteúdo. $<?xmlversion = "1.0"?>$

$< EXEMPLO >$

$< Estudante >$

$< Nome > NiviaMenezes < /Nome >$

$< Tel >$ 914.631.7722 $< /Tel >$

$< Fax >$ 914.631.7723 $< /Fax >$

$< E - Mail > nivia@bol.com < /E - Mail >$
$< /Estudante >$

$< /EXEMPLO >$

Entre as características do XML estão [K+02]:

- simplicidade de a comunicação - é uma linguagem auto-descritiva. Ou seja, a estrutura associada a um documento XML descreve os dados existentes, facilitando seu entendimento e a interpretação;
- flexibilidade - os dados podem ser armazenados e organizados de uma forma a customizar as necessidades de cada aplicação;
- padronização - o padrão do conjunto de caracteres para XML é o UNICODE (padrão internacional para textos). Por isso, várias linguagens e caracteres são suportadas, sendo também possível descrever os dados em qualquer linguagem;
- facilidade de transporte - o fato de ser baseado em texto torna o XML de fácil transporte entre sistemas e através da Internet
- facilidade de verificação e validação - a estrutura e a qualidade do documento podem ser verificadas, tornando possível validar o documento inteiro, a sintaxe e os tipos de dados antes dos dados serem processados pela aplicação. Logo, é possível adicionar uma detecção de erro mais completa e robusta dentro da aplicação;
- facilidade de combinação - a XML pode facilmente ser misturada com estilos (*stylesheets*) para criar várias saídas diferentes. O estilo serve para permitir que os mesmos dados sejam utilizados de formas diferentes, apenas modificando o estilo.
- facilidade de representação - virtualmente, qualquer tipo de dados pode ser expresso como um documento XML. A XML somente fornece as regras que dizem como descrever os dados.
- legibilidade por humanos - como a XML foi projetada para ser textual, qualquer pessoa pode ler e descobrir seu conteúdo. Tal fato não acontece com arquivos binários;
- suporte pela indústria - um grande número de ferramentas está sendo fornecida pelos navegadores, banco de dados e sistemas operacionais, facilitando a adoção, com um baixo custo, da XML em empresas de pequeno e médio porte para a importação e exportação;
- suporte pelos banco de dados relacionais: tais bancos de dados possuem capacidade nativa para ler e gerar dados XML;
- suporte pelas tecnologias disponíveis: Um grande família de tecnologias de suporte ao XML está disponível para a interpretação e transformação de dados XML para apresentação em páginas Web e geração de relatório;

A grande vantagem da XML é a simplicidade. E foi prezando por isso que a linguagem não fornece alguns recursos, tais como:

- suporte à apresentação de dados - ao contrário do HTML, XML não assume o papel de como as etiquetas serão desenhadas em um browser ou dispositivo gráfico;
- tipos de dados construídos no XML - para tanto, os DTDs e Esquemas XML [W3S03b] fornecem suporte para definição de estruturas e tipos de dados associados ao documento XML;
- recursos de transporte - a especificação XML não faz qualquer suposição sobre como o XML será transportado na Internet. Por isso, existem vários protocolos que podem fazer seu transporte como HTTP, FTP, SMTP.

3.2.2 Padrões da estrutura do XML

A seguir são descritas algumas características para a estrutura de um documento XML:

- a XML permite que os dados sejam armazenados como elemento e atributos;
- os elementos e atributos podem ter nomes que dêem significado aos dados;
- os elementos são definidos com etiquetas iniciais e finais que são a base para as representações estruturadas de documentos;
- os elementos podem conter dados textuais ou outros elementos;

A XML fornece um grande poder para criar linguagem e elementos de marcação customizados. Essa flexibilidade poderia expressar o caos nos analisadores gramaticais (*parsers*) caso eles não tivessem regras referentes à marcação. Os documentos que seguem todas as regras de sintaxe XML são referenciados como documentos bem formados. Um documento bem-formado tem:

- um único elemento raiz;
- etiquetas corretamente aninhadas;
- etiquetas corretamente fechadas;
- valores de atributos dentro das aspas; e
- um valor apenas por atributo.

Além de ser bem-formado, o documento XML deve ser válido. Um documento é considerado válido se estiver de acordo com o DTD [W3S03a] ou esquema escrito para descrever sua estrutura e seus requisitos. A seguir serão descritos alguns aspectos da XML.

3.2.2.1 Etiquetas e Elementos

A marcação do XML consiste de etiquetas, como exemplo tem-se $< Estudante >$. O autor de um arquivo XML cria etiquetas para descrever os dados que o arquivo possui. Essas etiquetas são semelhantes às etiquetas do HTML. Os elementos são a base para os documentos XML. Existem três termos usados referentes às etiquetas. Os termos são etiqueta aberta, $< Estudante >$, etiqueta fechada, $< /Estudante >$, e etiqueta vazia. Uma etiqueta aberta é a primeira etiqueta do par. A etiqueta fechada é a etiqueta final do par. Já a etiqueta vazia é uma etiqueta sozinha sem dados associados.

As etiquetas são usadas para construir os elementos. Um elemento consiste de uma etiqueta aberta, uma etiqueta fechada e tudo entre elas. A seguir é mostrado um exemplo de elemento: $< Estudante >< /Estudante >$. Os elementos podem conter outros elementos, dados ou textos, ou podem ser vazios.
$< Estudante >$

$< Nome > NiviaMenezes < /Nome >$
$< /Estudante >$

A seguir serão apresentadas algumas regras para o uso de etiquetas [K+02]:

- as etiqueta são case-sensitive. Isto significa que $< NOME >$ e $< Nome >$ são etiquetas diferentes;
- não é permitido espaços em branco no início da etiqueta, mas é permitido no final;
- o nome da etiqueta deve começar com uma letra ou sublinhado;
- o nome da etiqueta pode conter letras, números, hífens, pontos e sublinhados;
- toda etiqueta aberta deve ser fechada;
- todo elemento deve ser corretamente aninhado antes que uma nova etiqueta seja aberta, pois a XML possui regras estritas para aninhamento;

Todos os documentos XML possuem algumas características comuns, tais como Elemento Raiz, Atributos, que serão descritos nas próximas seções.

3.2.2.2 Elemento Raiz

Todos os documentos XML devem ter um único elemento raiz, também conhecido como o elemento do documento. Esse elemento, que é definido pelo autor, contém o resto do documento XML. Apenas dois tipos de declarações podem estar fora do elemento raiz: Uma declaração do documento (que é sempre a primeira linha do documento XML) e as instruções de processamento. O elemento raiz do exemplo apresentado anteriormente é $< EXEMPLO >$. É importante observar que $< EXEMPLO >$ é a primeira etiqueta que não é uma declaração XML nem uma etiqueta de processamento, e é fechada pela etiqueta $< /EXEMPLO >$ no final, além disso, todas as outras etiquetas estão dentro do elemento raiz.

3.2.2.3 Atributos

Os atributos constituem uma outra parte importante dos documentos XML. Um atributo é um par nome/valor que pode ser encontrado em uma etiqueta aberta, eles fornecem informações adicionais a um elemento particular. A seguir é apresentado um exemplo. $< Nome > NiviaMenezes < /Nome > < Tel >$ 922334455 $< /Tel >$

Os atributos são úteis para fornecer as propriedades de um elemento.

A seguir são apresentadas as regras para atributos:

- qualquer dado que precise ser mostrado deve ser armazenado como elemento
- qualquer dado cujo objetivo é modificar a forma da apresentação de um elemento deve ser armazenado como um atributo;
- os atributos consistem de um nome da propriedade, um sinal de igualdade, e o valor da propriedade entre aspas;
- o nome da propriedade é case sensitive;
- não pode existir duas propriedades com o mesmo nome na mesma etiqueta;
- pode existir mais de um atributo por etiqueta;
- deve existir aspas para o valor de um atributo. Podem ser usadas aspas simples ou duplas.

Todo arquivo XML deve ter uma estrutura válida. Para validar os documentos XML foi definido o DTD (*Document Type Definition*), que será descrito na próxima seção.

3.2.3 DTD - *Document Type Definition*

Um DTD é a definição formal da estrutura de um determinado documento XML e está normalmente armazenado num arquivo. Construir um documento XML de acordo com um determinado DTD significa que esse documento está em conformidade e a sua estrutura é válida para esse DTD [COY00].

No DTD, definem-se os nomes dos elementos e atributos do documento, além das suas características (a sua ordenação, posição relativa e conteúdo). Um documento XML bem formatado deve conter a localização do DTD ao qual está associado. Um DTD define regras para a especificação de uma classe de documentos, tais como:

- que tipos de elementos podem existir em um documento;
- que atributos esses elementos podem ter; e
- como as instâncias desses elementos estão hierarquicamente relacionadas.

A estrutura especificada em um DTD [W3S03a], segundo sua definição no padrão SGML (*Standard Generalized Markup Language*), possui uma propriedade importante: apenas a estrutura lógica de um documento é descrita, não sendo fornecida informação alguma sobre a semântica da apresentação do documento.

Um DTD pode ser referenciado interna ou externamente a um documento XML

Segundo a proposta do DTD, todo documento XML é composto pelos blocos elementos, etiquetas, atributos, entidades, PCDATA e CDATA [W3S03a]:

Além do DTD, existe o esquema XML que também é uma forma de validar um arquivo XML, com características mais flexíveis que o DTD. A próxima seção descreve os esquemas XML com mais detalhes.

3.2.4 Esquema XML

Assim como o DTD, o Esquema XML [W3S03b] propõe a definição dos blocos de construção válidos para um documento XML. Um Esquema XML é um linguagem para descrição de documentos, que permite especificar, de maneira mais precisa que o DTD, como um documento XML deve ser formatado.

Portanto, o Esquema XML define os seguintes aspectos [W3S03b]:

- os elementos que podem aparecer em um documento;
- os atributos que podem aparecer em um documento;

- os elementos que podem ser elementos-filhos;
- a ordem dos elementos-filhos;
- o número de elementos-filhos;
- os tipos de dados para elementos e atributos;
- valores fixos e padrão para os elementos e atributos.

Segundo [W3S03b], os Esquemas XML possuem várias características que o diferenciam e tornam-os melhores do que os DTS, entre elas estão:

- habilidade de extensão para adições futuras;
- mais recursos e utilidade do que os DTSs, visto que suportam tipos de dados e namespace;
- escrita em XML são escritos em XML; e
- suporte a tipos de dados.

Com o suporte a tipos de dados, os Esquemas XML ganham muita força, pois torna-se mais fácil executar atividades como:

- descrever o conteúdo de documentos;
- validar a corretude dos dados;
- trabalhar com dados de um banco de dados;
- definir restrições sobre os dados;
- definir padrões e formato para os dados; e
- converter dados entre diferentes tipos de dados.

Para transportar os arquivos XML, faz-se necessário um protocolo para permitir a comunicação entre os fornecedores e os clientes de Web Services, a seguir será descrito o protocolo SOAP e suas características.

3.3 SOAP - *Simple Object Access Protocol*

O SOAP é um protocolo elaborado para facilitar a chamada remota de funções via Internet, permitindo que dois programas se comuniquem de uma maneira tecnicamente muito semelhante à invocação de páginas Web [BOX00, NAK01]. Ele é uma especificação usada para descrever mensagens XML que podem ser enviadas através de uma rede. O protocolo SOAP localiza-se acima do protocolo de transporte (HTTP, SMTP etc.), usado para enviar mensagens, e abaixo dos documentos XML de domínio específico, como mostra a Figura 3.2 [W+02].

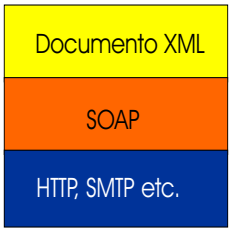

Figura 3.2. *Pilha de Protocolos*

O maior objetivo do SOAP é a simplicidade e extensibilidade, isso significa que existem muitas características de sistemas de mensagem tradicionais e sistemas de objetos distribuídos que não são partes do núcleo da especificação SOAP. Entre elas estão coleta de lixo distribuída, objetos por referência, ativação e *batching* de mensagem [BOX00].

O SOAP fornece um mecanismo simples e leve para troca de informações tipadas e estruturadas entre aplicações, em um ambiente descentralizado e distribuído usando XML [S+01]. O SOAP não define qualquer semântica de aplicação como um modelo de programação ou semântica específica de implementação, ao invés, define um mecanismo simples para expressar semântica fornecendo um modelo de empacotamento modular e mecanismos de codificação para dados dentro dos módulos. Isso permite que o SOAP seja usado em uma grande variedade de sistemas abrangendo desde o sistema de mensagem ao RPC (*Remote Procedure Call*) [BOX00].

O SOAP consiste de três partes [BOX00]:

- o Encapsulador do SOAP que define um ambiente global para expressar o que está na mensagem, quem deveria lidar com ela e se ela é opcional ou obrigatória;
- as regras de codificação que definem um mecanismo de serialização que pode ser usado para trocar instâncias de tipos de dados definidos na aplicação; e
- a representação RPC (*Remote Procedure Call*) que define uma convenção que pode ser usada para representar chamadas e respostas a procedimentos remotos.

Embora essas partes sejam descritas juntas como parte do SOAP, elas são funcionalmente ortogonais. Em particular, o encapsulador e as regras de codificação são definidos em *namespaces* diferentes para promover simplicidade através da modularidade [W3S02].

3.3.1 O Modelo de Troca de Mensagem SOAP

As mensagens SOAP são fundamentalmente transmissões de uma via (SIMPLEX), do remetente para o receptor, mas as mensagens SOAP são geralmente combinadas para implementar padrões tais como pedido/resposta.

Uma mensagem SOAP é um documento XML que consiste de um encapsulador SOAP obrigatório, um cabeçalho SOAP opcional e um corpo SOAP obrigatório. Esse documento XML é referenciado como uma mensagem SOAP pelo resto dessa especificação. O identificador do *namespace* para atributos e elementos nessa seção é
"*http://schemas.xmlsoap.org/soap/envelope/*".

Uma mensagem SOAP contém [BOX00]:

- o Encapsulador é o elemento do topo do documento XML representando a mensagem;
- o cabeçalho é um mecanismo genérico para adicionar características à mensagem SOAP de uma maneira descentralizada sem acordo prévio entre as partes comunicantes. O SOAP define alguns atributos que podem ser usados para indicar quem deve lidar com uma característica e se é opcional ou obrigatória; e
- o corpo é um local para uma informação obrigatória planejada para o último receptor da mensagem. O SOAP define um elemento para o corpo, usado para reportar erros.

Uma aplicação SOAP recebendo uma mensagem SOAP deve processar a mensagem executando as ações, na seguinte ordem [BOX00]:

1) identificar todas as partes da mensagem SOAP planejadas para a aplicação;
2) verificar que todas as partes obrigatórias identificadas no passo 1 são suportadas pela aplicação para essa mensagem e processá-las adequadamente. Se não for esse o caso, então a mensagem deve ser descartada. O processador pode ignorar as partes opcionais identificadas no passo 1 sem afetar a saída do processamento; e
3) remover todas as partes identificadas no passo 1 antes de encaminhar a mensagem, se a aplicação SOAP não for o último destino da mensagem.

Processar uma mensagem ou parte da mensagem requer que o processador SOAP entenda, entre outras coisas, o padrão de troca usado (em um sentido, pedido/resposta, *multicast*, etc.), o papel do receptor no padrão, o emprego dos mecanismos RPC (*Remote Procedure Call*), a representação e codificação dos dados, bem como outras semânticas necessárias para o processamento correto.

A seguir serão descritos mais detalhes sobre as partes que compõem uma mensagem SOAP.

3.3.2 Encapsulador SOAP

O elemento encapsulador, também conhecido como *envelope*, está localizado no topo da hierarquia do protocolo SOAP. Ele é composto de dois elementos: *XMLnamespace* e *encodingStyle*. O XMLnamespace é um conjunto de nomes para tipos de elementos XML e nomes de atributos, isto é, um esquema XML [BHA00]. Um dos esquemas mais usados nas mensagens SOAP encontra-se em http://schemas.xmlsoap.org/soap/envelope. O atributo encodingStyle identifica o tipo de dados reconhecido pelas mensagens SOAP, além de especificar como os dados devem ser serializados para o transporte através da web. É possível indicar mais de um URI para identificar as regras de serialização, onde a ordem de colocação é da mais específica para a menos específica.

3.3.3 Cabeçalho SOAP

O SOAP fornece um mecanismo flexível para estender uma mensagem de uma maneira descentralizada e modular sem conhecimento prévio entre as partes comunicantes. Exemplos típicos de extensões que podem ser implementadas como entradas de cabeçalho são autenticação e gerenciamento de transações etc.

O elemento cabeçalho é codificado como o primeiro elemento do encapsulador XML do SOAP.

As regras de codificação para as entradas do cabeçalho são as seguintes:

- uma entrada de cabeçalho é identificada pelo nome completo de seu elemento qualificado, que consiste de *namespace* URI e o nome local. Todos os elementos imediatos do cabeçalho do SOAP devem ser *namespace* qualificado;
- o atributo *encodingStyle* do SOAP deve ser usado para indicar o estilo de codificação usado para as entradas do cabeçalho; e
- o atributo *mustUnderstand* do SOAP e o atributo ator do SOAP pode ser usado para indicar como processar a entrada e por quem.

3.3.4 Corpo SOAP

O elemento Corpo do SOAP fornece um mecanismo simples para troca de informação obrigatória planejada para o último receptor da mensagem. Os usos típicos do elemento Corpo incluem ordenação de chamadas RPC e relatório de erros. O elemento Corpo é codificado como um elemento imediato do elemento Encapsulador XML do SOAP.

Se um elemento Cabeçalho estiver presente, então o elemento Corpo deve imediatamente seguir o elemento Cabeçalho, por outro lado ele deve ser o primeiro elemento imediato do elemento Encapsulador.

Todos os elementos imediatos primários do elemento Corpo são chamados de entradas do Corpo e cada entrada é codificada como um elemento independente dentro do elemento Corpo do SOAP.

As regras de codificação para as entradas do Corpo são as seguintes:

- uma entrada Corpo é identificada pelo nome do elemento totalmente qualificado, que consiste de *namespace* URI e o nome local. Os elementos imediatos do Corpo do SOAP podem ser *namespace* qualificado;
- o atributo *encodingStyle* do SOAP deve ser usado para indicar o estilo de codificação usado para as entradas do corpo;
- o SOAP define uma entrada do corpo, que é entrada Fault usada para reportar erros.

3.3.5 Relação entre o Cabeçalho e Corpo SOAP

Apesar do Cabeçalho e o Corpo serem definidos como elementos independentes, eles estão relacionados. A Figura 3.3 ilustra uma mensagem SOAP. Ou seja, a relação entre uma entrada Cabeçalho e uma entrada Corpo é a seguinte: uma entrada de Corpo é semanticamente equivalente a uma entrada cabeçalho.

Além do protocolo de comunicação entre os Web Services, existe uma linguagem para descrever os serviços, seus parâmetros de entrada e resultados fornecidos, tal linguagem é conhecida como WSDL, que será descrita na próxima seção.

3.4 WSDL - *Web Service Description Language*

A WSDL [C+03] é uma linguagem que fornece um modelo e um formato XML para descrição de Web services. A WSDL permite a separação entre a descrição da funcionalidade abstrata oferecida pelo serviço dos detalhes concretos de uma descrição de serviço tais como "como" e "onde" a funcionalidade é oferecida [C+03].

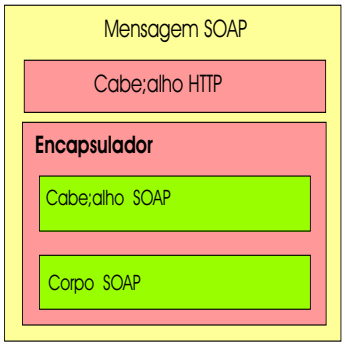

Figura 3.3. *Formato de uma Mensagem SOAP*

Um documento WSDL serve como descrição da linguagem e da plataforma de um ou mais serviços, descrevendo-os, como acessá-los e o tipo de resposta fornecida (quando existir). Esse documento pode ser trocado privativamente ou armazenado em um registro UDDI (que também pode ser público ou privado) para permitir o acesso externo. A WSDL oferece um formato de interface neutro e padronizado, o que permite expor uma interface Java, documentação baseada em texto ou qualquer outro tipo de interface para o serviço. A WSDL é baseada no formato XML para descrição de tipos, mensagens, operações, interfaces (chamadas de *PortTypes*), localizações e ligações de protocolos. Os Web services são descritos como um conjunto de mensagens e operações com pontos de marcação [W+02].

Na WSDL, as mensagens descrevem a comunicação entre cliente e serviço, descrito pelos tipos de dados trocados. As operações consistem de mensagem de entrada e saída. As *PortTypes* consistem de uma coleção de operações, elas são a fronteira para certos protocolos (esta ação é chamada de ligação). A WSDL suporta ligações com os protocolos SOAP 1.1, HTTP GET/POST, a MIME, no entanto, é possível extender e pode ser usado com outros protocolos de rede e formatos de mensagem.

Na Figura 3.4 é apresentada a estrutura de um documento WSDL:

3.4.1 Modelo de Componentes

Segundo [C+03], o modelo conceitual de componentes do WSDL pode ser descrito como um conjunto de componentes e suas propriedades. Nesta seção será apresentada uma visão geral dos componentes existentes no WSDL. Dentre os componentes existentes no modelo WSDL,pode-se citar os seguintes [C+03]:

- componentes de definições;
- componente de mensagens;
- componentes de divisão;

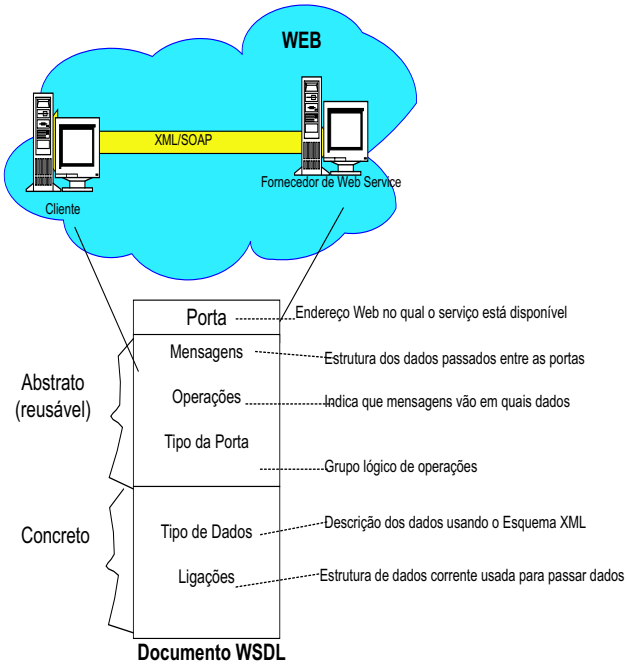

Figura 3.4. *Estrutura de Documento WSDL*

- componentes de esquema;
- componentes de tipo de porta;
- componentes de operação de tipo de porta;
- componentes de referência de mensagem;
- componente de ligação;
- componentes da operação de ligação;
- componentes de referência da mensagem de ligação;
- componentes de serviço;
- componentes de porta;

Para que os fornecedores de Web Services disponibilizem seus serviços, faz-se utilização do UDDI que permite a publicação e descoberta de informações desses serviços. As características do UDDI serão descritas na próxima seção.

3.5 UDDI - *Universal Description Discovery and Integration*

Segundo [Sec00] e [W+02], o UDDI é uma especificação que fornece recursos para publicação e descoberta de informações, baseadas na Web, sobre serviços, tais como documentos WSDL, interfaces Java padrão e documentos XML. Foi lançado pela IBM, Microsoft e Ariba em 2000, seu modelo de dados e sua API de programação são baseadas no XML e no SOAP, o que fornece uma maneira para publicar e localizar qualquer tipo de serviço.

Pode-se fazer uma analogia entre o UDDI e uma máquina de busca da Internet, pois ambos disponibilizam as informações por índices e por categoria. No entanto, o UDDI traz não somente a localização do serviço como também fornece informações sobre o serviço, tais como o seu funcionamento, os parâmetros esperados e os valores retornados.

É importante saber que, apesar da especificação do UDDI utilizar o SOAP como protocolo de comunicação, é possivel utilizar outros protocolos tais como o SOAP baseado em Java ou uma API cliente para UDDI baseada em Java [IBM02]. A arquitetura do UDDI será descrita na próxima seção.

3.5.1 Arquitetura UDDI

Para acessar os serviços UDDI, um cliente envia uma mensagem SOAP codificada de acordo com o estilo de codificação UDDI. A arquitetura básica de consulta a um registro UDDI é mostrada na Figura 3.5. Um pedido SOAP é enviado ao servidor e desserializado pelo processador SOAP no nó de registro UDDI. As chamada UDDI são feitas em frente ao serviço de registro que segue as consultas ao banco de dados UDDI. Uma resposta é devolvida através do processador SOAP para serialização e então remetida para o servidor Web para ser retornada para a aplicação cliente.

Como mostrado na Figura 3.5, o uso do UDDI exige um nó de registro e um cliente capaz de enviar pedidos UDDI baseados em SOAP. Os registro podem ser privados (criados para *intranets* corporativas ou para *extranets Business-Business*) ou públicos (como exemplo, tem-se ibm.com, microsoft.com, uddi.org, e xmethods.com). Os registros reconhecem o UDDI como uma forma padrão para registrar e localizar serviços comerciais dentro de uma intranet corporativa ou como uma forma padronizada de disponibilizar um registro de serviço para seus parceiros.

Um nó UDDI serve como fornecedor de serviço, um registro de serviço e um solicitante de serviço. Um fornecedor de serviço porque publica os serviços de e-business. Um registro de serviço por disponibilizar um diretório navegável de Web Services. E um solicitante por localizar um serviço solicitado e ligar o cliente ao serviço. Para as tarefas de registro

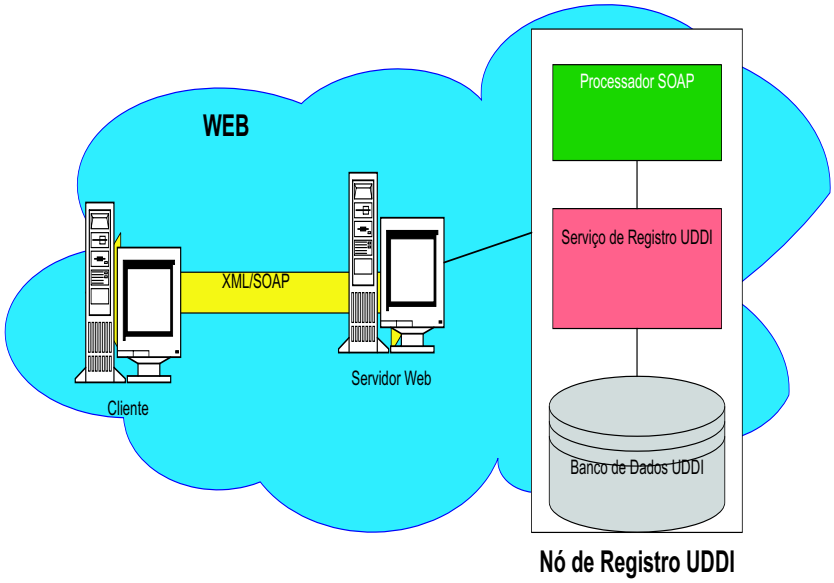

Figura 3.5. *Arquitetura UDDI*

e localização, os comando UDDI são enviados no corpo de uma mensagem SOAP para um registro UDDI [W+02].

O principal componente do projeto UDDI é o registro UDDI do negócio, um arquivo XML usado para descrever as entidades do negócio e seus Web services. Conceitualmente, esse arquivo consiste de três componentes[Sec00]:

- Páginas Brancas - incluindo endereço, contato e identificadores conhecidos;
- Páginas Amarelas - incluindo categorias industriais baseadas no padrão de taxonomias; e
- Páginas Verdes - as informações técnicas sobre os serviços disponibilizados pelo negócio.

A seguir serão descritos alguns detalhes sobre os componentes de Páginas Brancas, Amarelas e Verdes.

3.5.1.1 Componentes UDDI

Como dito anteriormente, os componentes do arquivo XML do UDDI é composto por três componentes: Páginas Brancas, Amarelas e Verdes [IBM02].

Páginas Brancas

As paginas brancas contém informações sobre o negócio, incluindo o nome, detalhes sobre contatos e a localização do negócio. Além disso, a identificação do negócio pode ser feita com a utilização de identificadores únicos como IDs. Ao inserir essa informação no registro, um cliente é capaz de procurar o registro baseado na informação sobre o negócio.

As informações das páginas brancas são facilitadas através do elemento *businessEntity*.

Páginas Amarelas

Essas páginas contém informações por categoria sobre os serviços fornecidos por um negócio. A categorização é feita através da atribuição de uma ou mais taxonomias do negócio. Ou seja, o mesmo serviço pode pertencer a duas categorias dependendo das suas funcionalidades e da categorização adotada.

As informações das páginas amarelas, assim como das páginas brancas, são associadas ao elemento *businessEntity*.

Páginas Verdes

As paginas verdes contém informações técnicas sobre os serviços oferecidos pelo negócio. As informações incluem referem-se a todas as informações técnicas necessárias para utilização do serviço, tais como localização, categoria e a especificação para o serviço podem ser encontradas nas Páginas Verdes.

Incluem referência para as especificações dos Web Services, bem como suporte para ponteiros para vários arquivos e URL baseado em mecanismo de descoberta, se necessário.

As informações das páginas verdes são associadas através dos elementos *businessService* e *bindingTemplate*.

Para permitir a interação de um programa como registros, foi definifa a especificação UDDI, que será descrita a seguir.

3.5.1.2 Especificações UDDI

A estrutura UDDI consiste de muitas especificações que descrevem como um programa pode interagir com um registro, incluindo:

- **especificação da API do Programador UDDI -** define aproximadamente 30 mensagens SOAP que são usadas para realizar funções de investigação e publicação. Esta especificação descreve os detalhes de cada estrutura XML associada com esses mensagens; e
- **especificação da Estrutura de Dados UDDI -** define as quatro maiores estruturas usadas na API de programação, incluindo *businessEntity*, *businessService*, *bindingTemplate*, e *tModel*.

3.5.1.3 Modelo de Invocação UDDI

Cada Web Service publicado é modelado em uma estrutura *bindingTemplate*. A invocação de um Web Service é tipicamente executada baseada nos dados armazenados na *bindingTemplate*. Os passos para utilização do UDDI é mostrado a seguir [COY00]:

1) o programador, que deseja escrever um programa para utilizar um Web Service remoto, usa o registro de negócio do UDDI (via interface Web ou outra ferramenta que usa a API *Inquiry*) para localizar as informações do *businessEntity* registradas para ou pelo parceiro de negócio apropriado que está anunciando o Web service;

2) o programador solicita mais detalhes sobre o *businessService* ou solicita a estrutura completa do *businessEntity*. Como as estruturas *businessEntity* contém todas as informações sobre os Web Service publicados, o programador seleciona um bindingTemplate particular e salva-o em outro lugar para uso posterior.

3) o programador prepara o programa baseado no conhecimento da especificação para o Web Service. Essa informação pode ser obtida usando a informação da chave *tModel* contida no *bindingTemplate* para um serviço.

4) o programa invoca, em tempo de execução, o Web Service como planejado usando as informações do *bindingTemplate*. No caso geral, assumindo-se que um Web service remoto e a chamada de programa implementam precisamente as convenções para as interfaces requeridas, as chamadas para serviços remotos serão executadas com sucesso. Posteriormente, será abordado o tratamento e recuperação de falhas.

Considerando que as aplicações distribuídas possuem mais probabilidade de apresetarem falhas, é necessário que as mesmas sejam tratadas. A seguir serão descritos os mecanismos de tratamento de falhas e recuperação disponíveis no UDDI.

3.5.2 Falha e Recuperação no UDDI

Sabe-se que aplicações distribuídas têm naturalmente mais pontos de falhas do que as aplicações stand-alone, por isso, é importante para os clientes serem capazes de detectar e recuperar-se de falhas que ocorrem durante a interação com parceiros remotos. Vislumbrando essa realidade, o UDDI possui recursos para tratar e recuperar falhas.

Os aspectos referentes à qualidade de serviço do UDDI são abordados através da definição de convenção de chamadas que envolve o uso do elemento *bindingTemplates*. Ao ocorrer uma falha, a informação armazenada é atualizada de acordo com a informação corrente no registro Web UDDI [COY00].

A seguir é apresentado um cenário que descreve com uma recuperação de erros é feita no Web Services [COY00]:

1) o desenvolvedor prepara um programa para usar um Web Service, ocultando o *bindingTemplates* apropriado para ser usado em tempo de execução;

2) durante a execução do programa, o programa que chama o Web service remoto o *bindingTemplates* que foi anteriormente obtido de um registro Web UDDI;

3) se a chamada falhar, o programa usa o valor *bindingKey* e a faz uma chamada à API *getbindingTemplate* para adquirir uma nova cópia de um *bindingTemplates* para esse único Web Service; e

4) o programa compara as novas informações do *bindingTemplate* com as anteriormente armazenadas, se elas forem diferentes, faz novamente a chamada usando o novo *bindingTemplate*; Se as versões são iguais, o cliente faz novamente a chamada. Esta abordagem, chamada de nova tentativa para falha (*retry on failure*), é mais eficiente do que adquirir uma nova cópia dos dados *bindingTemplate* antes de cada chamada. Além de ser útil quando um negócio precisa redirecionar o tráfego para um novo site, pois é necessário apenas ativar o novo site e mudar a informação de localização publicada para cada *bindingTemplate* afetado.

Os Web Services podem ser implementados em diversas plataformas, pois os mesmos são uma especificação. No caso deste trabalho, será utilizada a plataforma J2EE, haja vista que um dos objetivos do trabalho é avaliar os recursos de tal plataforma para implementaçàо de Web Service.

Capítulo 4

J2EE - Java 2 Enterprise Edition

Tendo em vista que a proposta deste trabalho é a utilização da plataforma J2EE [MIC02a] para integração de ambientes heterogêneos por meio de Web Services, faz-se necessária a descrição das particularidades da mesma. Por isso, neste capítulo serão apresentadas a descrição geral, a arquitetura e as características dos componentes da plataforma J2EE, bem como as APIs Java para processamento XML.

4.1 Introdução

Uma plataforma é uma combinação de hardware e software necessários para executar uma aplicação [Wor01]. De forma geral, a plataforma J2EE é uma coleção de especificações de tecnologias relacionadas que descrevem APIs (*Application Programming Interface*) e políticas requeridas. Essa plataforma representa o consenso de fornecedores de softwares comerciais envolvidos com quais facilidades uma plataforma deveria prover e como acessá-las. Os fornecedores competem na implementação de uma especificação comum, fornecendo aos clientes a liberdade para escolher a tecnologia que mais os agrada e adequa-se às suas necessidades e orçamento, e trocar de fornecedor se as necessidades ou o orçamento mudarem [Wor01].

A plataforma J2EE oferece um modelo de aplicação distribuída, componentes reusáveis, um modelo de segurança unificado, controle de transações flexível e suporte a Web Services utilizando a troca de dados em XML, que é baseada em padrões e protocolos.

Para entender a plataforma J2EE, é necessário saber um pouco da suas origens, ou seja, conhecer a linguagem Java. Para tanto, a seguir serão apresentadas as características da linguagem Java.

4.2 Java: A Base do J2EE

A linguagem de programação Java foi criada pela Sun Microsystems [MIC98], com o objetivo de solucionar os problemas da Internet relativos à grande diversidade de hardware, sistemas operacionais e servidores web. O seu primeiro grande passo veio com a interatividade fornecida pelos *applets* (mini-programas que rodam dentro de um browser e que são baixados automaticamente para o computador do usuário quando requisitamos uma página da web) [FLA99].

Atualmente, Java vem sendo utilizada nos mais variados dispositivos, incluindo computadores de bordo para carros, televisores, telefones e *SmartCards*, entre outros [Wor01]. Essa linguagem teve sua origem em um grande projeto para desenvolver softwares para produtos eletrônicos, sendo que seu objetivo principal é resolver um grande número de problemas nas práticas de programação modernas. Ela possui algumas características importantes, tais como: simplicidade, orientação a objeto e dinamismo, capacidade de distribuição, segurança, portabilidade, além da sincronização.

Para oferecer simplicidade, procurou-se construir uma linguagem que pudesse ser implementada facilmente sem a necessidade de um treinamento extensivo. Outro aspecto da simplicidade é o tamanho. A linguagem Java é pequena, pois um dos seus objetivos é habilitar a construção de software que possa ser executado automaticamente, sem a necessidade de sistema operacional, bibliotecas etc.[G$^+$97].

Quanto à orientação a objetos, os recursos oferecidos pela linguagem é a capacidade de facilitar a clara definição de interfaces, classes, objetos, herança e encapsulamento, possibilitando o desenvolvimento de softwares reutilizáveis e um desenvolvimento dinâmico de aplicações [RIC01].

A característica de distribuição pode ser observada devido à biblioteca extensa de rotinas para serem copiadas facilmente com protocolos TCP/IP como HTTP e FTP. Aplicações Java podem abrir e acessar objetos através da rede via URL's e com as mesma facilidades que os programadores usam quando acessam um sistema de arquivo local.

A linguagem Java é destinada para escrever programas que possam ser confiáveis em uma variedade de hipóteses. Por ser uma linguagem fortemente tipada a linguagem admite uma extensiva checagem em tempo de compilação. Além disso, a linguagem Java possui a propriedade de incorporar um *garbage collector*, ou seja, um mecanismo para o gerenciamento automático dinâmico de memória [L$^+$96]. Todas essas propriedades tornam a linguagem Java robusta.

Em virtude da linguagem ser destinada para utilização de aplicações em redes e meios distribuídos, muita ênfase tem sido colocada na questão da segurança. Java possibilita a construção de sistemas livre de vírus e livre de adulterações. As técnicas de autenticação são baseadas na criptografia de chave pública [BAR].

Como citado anteriormente, Java foi projetada para suportar aplicações em redes. Em geral, as redes são compostas de uma variedade de sistemas com uma variedade de CPU's e arquiteturas de sistemas operacionais. Para fazer com que uma aplicação Java seja executada em qualquer lugar na rede, o compilador gera código para uma máquina abstrata, a JVM (*Java Virtual Machine*) [MIC98]. Para portar código Java, basta ter uma versão da JVM funcionando no hardware especificado.

Como é sabido, as máquinas abstratas diminuem a perda semântica entre a linguagem fonte e o objeto, sendo sua migração para diversas plataformas de hardware feita de maneira simples. Pode-se observar o sucesso da neutralidade de arquitetura através da ampla disseminação de Java para possivelmente todas as plataformas de hardware existentes no mercado. [MIC98]

O fato de ser uma arquitetura neutra é um grande passo para ser portável, mas é necessário mais que isso. Visando a portabilidade, Java não possui aspectos de implementação dependentes da especificação, ou seja, os tamanhos dos tipos de dados primitivos são especificados. As bibliotecas, que são parte do sistema, definem interfaces portáveis [MIC98].

Existem muitas coisas acontecendo ao mesmo tempo no mundo a nossa volta. Por isso, é importante construir aplicações com múltiplos eventos. Isso é conseguido com o recurso de *multhreading* fornecido pela linguagem Java. Visto que a mesma possui um sofisticado conjunto de primitivas de sincronização que são baseadas amplamente no uso de monitor e no paradigma de variável de condição. Integrando estes conceitos dentro da linguagem eles tornam muito mais fácil o uso e a robustez.

4.3 Arquitetura do J2EE

A plataforma J2EE fornece um modelo de aplicação distribuída em multi-camada, o recurso de reutilização de componentes, troca de dados integrada baseada em XML, um modelo de segurança unificado e controle de transações flexível [MIC02b]. É possível não apenas entregar soluções inovadoras ao cliente, como fazê-las mais rapidamente. Suas soluções baseadas em componentes independentes da plataforma J2EE não estão amarradas aos produtos e API dos fabricantes, ou seja, fabricantes e clientes possuem liberdade para escolherem os produtos e componentes que melhor atendam aos seus requisitos de negócio e tecnológico [ROM01].

Usando um modelo distribuído de aplicação multi-camada, é possível dividir a lógica da aplicação em componentes de acordo com a função e os vários componentes da aplicação que pertencem à aplicação J2EE. A Figura 4.1 mostra duas aplicações J2EE multi-camadas dividida em camadas descritas abaixo.

- componentes da camada cliente são executados nas máquinas clientes;
- componentes da camada web são executados no servidor J2EE;
- componentes da camada de negócio são executados no servidor de aplicação J2EE;
- software da camada EIS (*Enterprise Information System*) é executado no servidor de EIS.

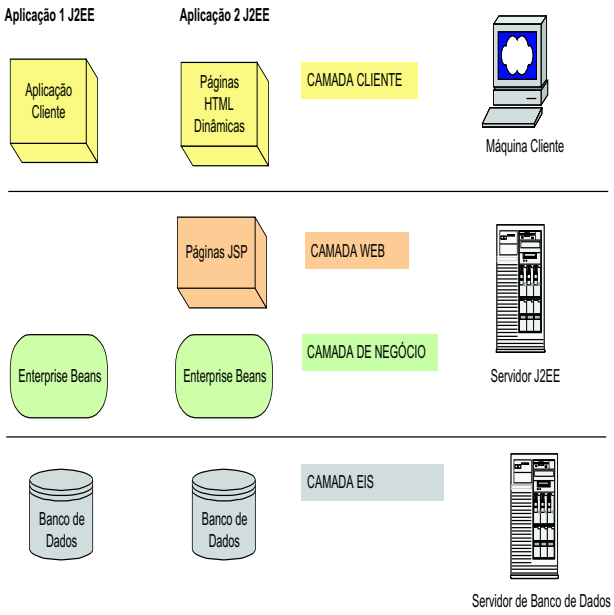

Figura 4.1. *Aplicações Multi-Camadas no J2EE*

As aplicações J2EE multi-camadas são geralmente consideradas como aplicações três camadas, pois elas são distribuídas sobre três diferentes localizações: máquinas-cliente, máquina servidor J2EE e a máquina de banco de dados. Aplicações com três camadas que executam desta maneira são extensões do modelo cliente servidor de duas camadas, colocando-se um servidor multi-threaded entre a aplicação cliente e o armazenamento do *back-end*.

Cada parte da arquitetura J2EE mostrada na Figura 4.1 será apresentada na próxima seção.

4.4 Componentes do J2EE

As aplicações J2EE são uma combinação de vários componentes. Um componente J2EE é uma unidade de software funcional auto-contido que é montada dentro de uma aplicação J2EE com classes e arquivos relacionados com outros componentes. A especificação J2EE define os seguintes componentes:[MIC02b]

- aplicações-clientes e applets que são componentes-clientes;
- os componentes *Java Servlet* e *JavaServer Pages* (JSP) que são componentes Web;
- componentes *Enterprise JavaBeans* (EJB) que são componentes de negócio; e
- aplicações multi-camadas.

Segundo [MIC02b], os componentes J2EE são escritos na linguagem de programação Java e compilados da mesma maneira que qualquer outro programa na linguagem. Quando se trabalha com a plataforma J2EE, a diferença é que os componentes são montados dentro da aplicação J2EE. Há uma verificação de que os componentes estão bem formados e compatíveis com a especificação J2EE e publicados (deployed) para produção onde eles são executados e gerenciados pelo servidor J2EE.

Os clientes J2EE serão descritos com mais detalhes na próxima seção.

4.5 Clientes J2EE

Uma aplicação J2EE pode ser baseada na web ou não. Uma aplicação cliente é executada na máquina cliente para aplicação não baseadas na web, e um navegadr web baixa as páginas web e applets para máquina cliente para aplicações J2EE baseadas na web. A seguir, são apresentados os tipos de componentes-clientes especificados na plataforma J2EE [MIC02b].

4.5.1 Aplicações-Cliente

Uma aplicação-cliente é executada na máquina cliente e provê uma maneira para os usuários manipularem as tarefas tais como sistema J2EE ou administração de aplicação. Ela tipicamente tem uma interface gráfica criada a partir das API's do Swing [MIC02b] ou Abstract Window Toolkit (AWT) [MIC02b], mas uma interface com linha de comando também é possível.

Os clientes podem ser clientes gordos, que são aplicações isoladas em um computador pessoal, e clientes magros, como as aplicações rodando em um navegador em um computador pessoal, aplicações rodando em assistentes digitais pessoais e até mesmo telefones celulares e outros dispositivos de comunicação pessoal [C$^+$01].

4.5.2 Applets

Uma página Web adquirida da camada web pode incluir um applet. Um applet é uma aplicação-cliente escrita na linguagem de programação Java que é executado na Java VM (*Virtual Machine*) instalada no navegador [D$^+$01]. Entretanto, sistemas-cliente necessitarão de *Java Plug-in* e possivelmente de um arquivo com política de segurança para que o applet possa ser executado com sucesso no navegador [MIC02b].

Applets executados em outros sistemas baseados em rede tais como periféricos portáteis ou telefone de carro podem manipular páginas *Wireless Markup Language* (WML) geradas por uma página JSP [K$^+$02] ou *servlet* [ARM03] executado no servidor J2EE. A página WML é entregue sobre o *Wireless Application Protocol* (WAP) e a configuração da

rede requer um *gateway* para traduzir WAP para HTTP e vice-versa. O *gateway* traduz o pedido WAP vindo de um periférico portável para um pedido HTTP para o servidor J2EE, e então traduz a resposta do servidor HTTP e da página WML para uma resposta do servidor WAP para mostrar no periférico portável.

4.5.3 Comunicação de Servidores J2EE

A Figura 4.2 mostra os vários elementos que podem compor a camada cliente. O cliente se comunica com a camada de negócio executada no servidor J2EE tanto diretamente, como também no caso de um cliente executado no browser, através de páginas JSP ou servlet executado na camada web.

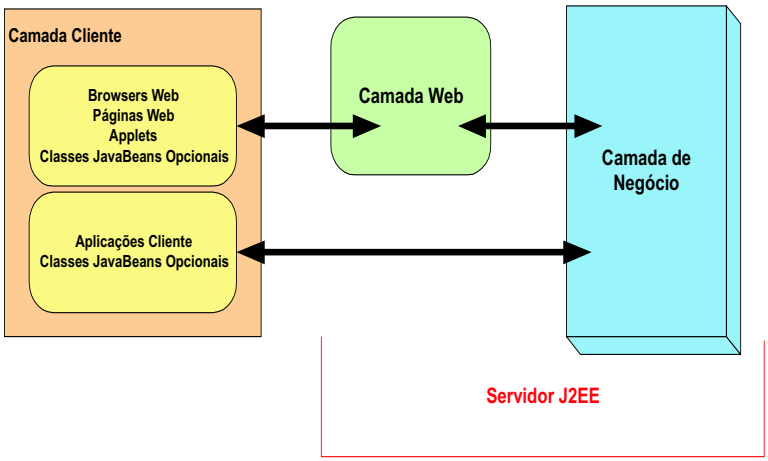

Figura 4.2. *Comunicação de Servidores*

4.6 Componentes Web

Segundo [MIC02b] os componentes web J2EE podem ser tanto páginas JSP ou servlets. Servlets são classes da linguagem de programação Java que dinamicamente solicitam e constroem respostas. Páginas JSP são documentos baseados em textos executados como servlets, mas permitem uma abordagem mais natural para criação de conteúdos estáticos.

As páginas estáticas HTML e os applets são empacotados com os componentes web durante a montagem da aplicação, mas não são considerados componentes web pela especificação J2EE. As classes de utilidades do lado do servidor também podem ser empacotadas com os componentes web , e como as páginas HTML, não são consideradas componentes web.

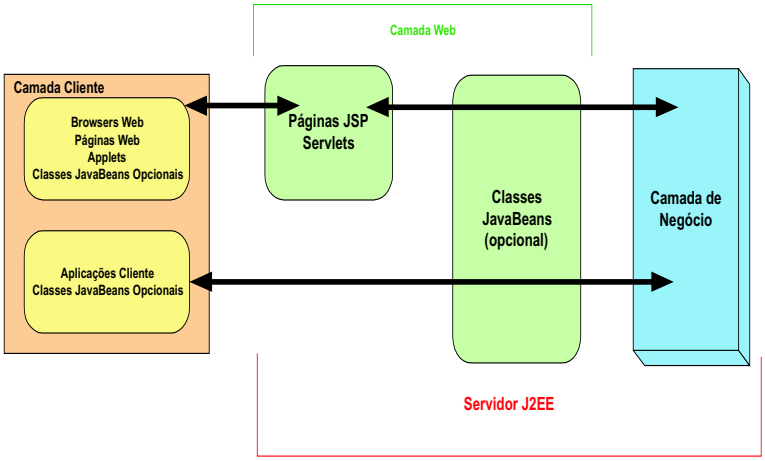

Figura 4.3. *Camada Web e Aplicação J2EE*

Como é mostrado na Figura 4.3, a camada web pode incluir um objeto *JavaBeans* para gerenciar a entrada de usuário e enviar essa entrada ao *beans* de negócio executado na camada de negócio, possibilitando o processamento [MIC02b].

4.7 Componentes de Negócio

O código de negócio resolve as necessidades de um domínio de negócio particular como um banco, varejo ou finança, é manipulado por um beans de negócio (*enterprise beans*) na camada de negócio. Figura 4.4 mostra como um *enterprise bean* recebe dados de programas cliente, processa-o (se necessário) e envia-o para camada de negócio do sistema de informação para armazenamento. Um *enterprise bean* também restaura dados do armazenamento, processa-o e envia de volta ao programa cliente.

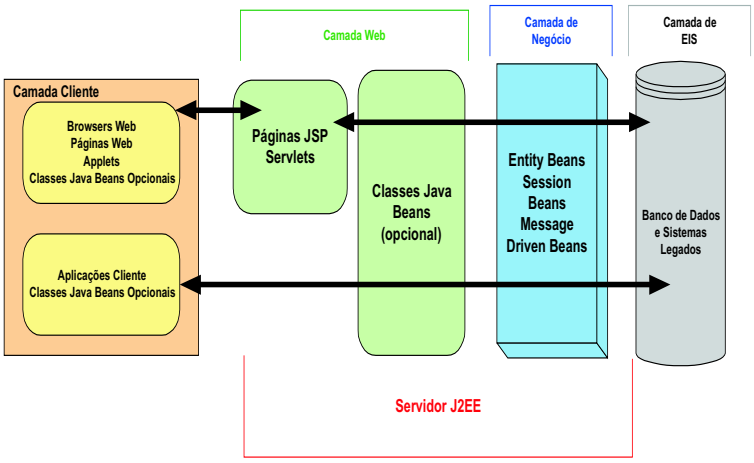

Figura 4.4. *Camadas de Negócio e EIS*

Existem três tipos de enterprise beans: *Session Beans*, *Entity Beans* e *Message Driven Beans* [MIC02b]. Um session bean representa uma conversação temporária com um cliente, ou seja, a finalização de uma conversa pelo cliente, encerra o ciclo de vida do *session bean* e seus dados. Ao contrário, um *entity bean* representa dados persistentes armazenados em uma linha da tabela do banco de dados. Se um cliente termina ou se o servidor é desligado, há garantias de que os dados do *entity bean* são salvos. Um *message-driven bean* combina as características do *session bean* e o *listener* de mensagem *Java Message Service* (JMS) [MIC02a], permitindo que um componente de negócio receba mensagens JMS de forma assíncrona.

4.8 Camada EIS (*Enterprise Information System*)

A camada EIS manipula o software de negócio do sistema de informação, e inclui os sistemas de infra-estrutura de negócio como Enterprise Resource Planning (ERP), processamento de transações de mainframe, sistema de banco de dados, e outros sistemas legados de informação. Os componentes das aplicações J2EE podem necessitar acessar a camada EIS para conectividade com banco de dados, por exemplo [MIC02b].

4.9 Containers J2EE

Normalmente, aplicações multi-camadas com cliente são difíceis de serem escritas por envolverem muitas linhas de código para manipular transações e gerenciamento de estado, *multithreading*, recurso de *pooling*, e outros detalhes complexos de baixo nível. A arquitetura J2EE baseada em componentes e independente de plataforma torna as aplicações J2EE fáceis de serem escritas porque a lógica do negócio é organizada em componentes reusáveis. Além disso, o servidor J2EE fornece serviços fundamentais na forma de um container para todos os tipos de componentes [MIC02b].

Os containers J2EE oferecem suporte para o modelo de programação de aplicação baseado em componente. Primeiramente, eles tornam automática grande parte da funcionalidade padrão que exige esforço de programação, tais como gerenciamento de transação e segurança [C$^+$01]. Além disso, eles oferecem APIs padronizadas na linguagem Java para outros recursos importantes para componentes, tais como mensagens, através do *Java Message Service*, e acesso a banco de dados, através da JDBC. Esses recursos do container unificam modelo de programação da J2EE, simplificando o desenvolvimento de aplicações, além de oferecerem suporte à portabilidade de componentes individuais e aplicações em larga escala.

A seguir são descritos os serviços e tipos de containers.

4.9.1 Serviços do Container

Containers são a interface entre um componente e a funcionalidade de baixo nível de uma plataforma específica que suporta o componente. Antes que um componente web, *enterprise bean* ou aplicação cliente possa ser executado, ele deve ser montado dentro da aplicação J2EE e publicado dentro do *container* [MIC02b].

O processo de montagem envolve a especificação da configuração do container para cada componente na aplicação J2EE e para aplicação J2EE propriamente dita. As configurações do container ajustam o suporte básico fornecido pelo servidor J2EE, que inclui serviços tais como segurança, gerenciamento de transações e *lookup Java Naming and Directory Interface* (JNDI), e conectividade remota. A seguir, são expostos alguns destaques [MIC02b]:

- o modelo de segurança J2EE permite a configuração de um componente web ou *enterprise bean* para que os recursos do sistema sejam acessados apenas por usuários autorizados;
- o modelo de transação J2EE permite a especificação de relacionamentos entre métodos que compõem uma transação simples para que todos os métodos em uma transação sejam tratados como uma unidade única;
- os serviços de *lookup* JNDI fornecem uma interface unificada para múltiplas nomenclaturas e serviços de diretório no negócio para que os componentes da aplicação possa acessar serviços de diretório e nomes; e
- o modelo de conectividade remota J2EE gerencia comunicação de baixo nível entre clientes e *enterprise beans*. Depois que um *enterprise bean* é criado, um cliente invoca métodos como se ele estivesse na mesma máquina virtual.

O fato da arquitetura J2EE fornecer serviços configuráveis significa que os componentes da aplicação dentro da mesma aplicação J2EE podem se comportar diferentemente, dependendo do local onde foi executado *deployment*, ou seja, a disponibilização ou publicação do componente. Por exemplo, um *enterprise bean* pode ter configurações de segurança que permitam-no um certo nível de acesso a dados em banco de dados em um único ambiente de produção e um outro nível de acesso a banco de dados em um outro ambiente de produção.

O *container* também gerencia serviços não-configuráveis como ciclo de vida de um *enterprise bean* e *servlets*, união dos recursos de conexão a banco de dados, persistência de dados e acesso a API's da plataforma J2EE. Embora a persistência de dados seja um serviço não-configurável, a arquitetura J2EE permite a substituição da persistência gerenciada pelo container (CMP - *Container-Managed Persistence*) pela persistência gerenciada pelo container (BMP - *Bean-Managed Persistence*), incluindo o código apropriado na implementação do enterprise bean quando se deseja mais controle do que a fornecida.

Por exemplo, é possível usar BMP para implementar métodos de busca ou criar uma cache padronizada de banco de dados.

4.9.2 Tipos de *Containers*

Segundo [MIC02b], o processo de *deployment* instala os componentes da aplicação J2EE nos *containers* como mostra a Figura 4.5.

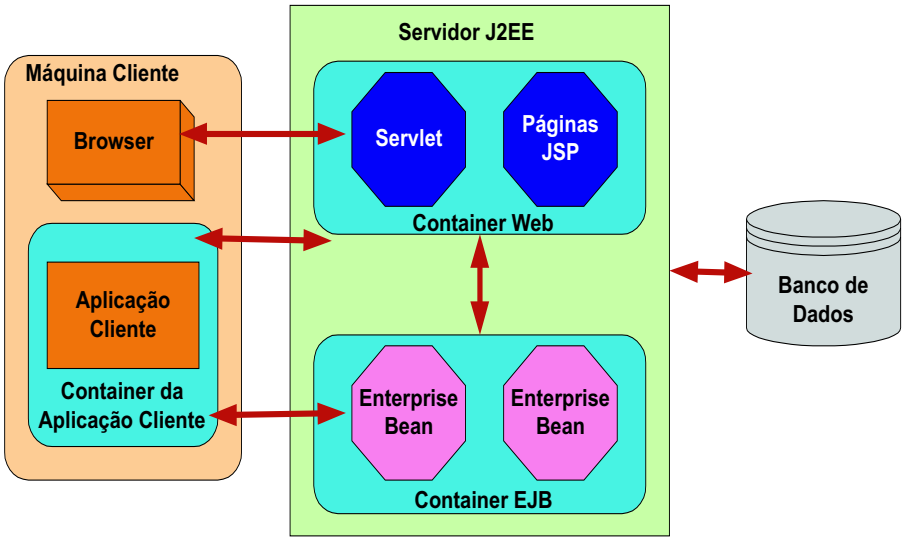

Figura 4.5. *Servidor e Containers J2EE*

- **servidor J2EE -** é a parte responsável por fornecer o ambiente de execução de um produto J2EE. Um servidor J2EE fornece EJB (*Enterprise JavaBeans*) e *containers* web;
- **container Enterprise JavaBeans (EJB) -** gerencia a execução de enterprise beans para aplicações J2EE. Enterprise beans e seus containers são executados no servidor J2EE;
- **container Web -** gerencia a execução dos componentes de páginas JSP e servlet para aplicações J2EE. Componentes web e seus containers são executados no servidor J2EE;
- **container Cliente da Aplicação -** gerencia a execução dos componentes clientes da aplicação. Aplicações clientes e seus containers são executados no cliente;
- **container Applet -** gerencia a execução de applets. Consiste de um navegador web e um Java Plug-in juntos executados no cliente.

4.10 Empacotamento

Os componentes J2EE são empacotados separadamente e colocados dentro de uma aplicação para publicação. Cada componente, seus arquivos relacionados como arquivos GIF e HTML ou classes de utilidades do lado do servidor, e um *Deployment Descriptor* (DD), são montados dentro de um módulo e adicionados à aplicação J2EE.

Uma aplicação J2EE é composta de um ou mais *enterprise bean*, ou módulos de componentes web ou aplicação-cliente. A solução final do negócio pode usar uma aplicação J2EE ou ser composta de duas ou mais aplicações J2EE dependendo dos requisitos de projeto [MIC02b]. Tal aplicação e cada um dos seus módulos tem seus próprios DD. Um DD é um documento XML com a extensão .xml que descreve as configurações do *deployment* do componente. Um DD de um módulo enterprise bean, por exemplo, declara atributos de transação e autorizações de segurança para um *enterprise bean*. Porque a informação do DD é declarativa, ela pode ser mudada sem modificar o código-fonte do bean. Em tempo de execução, o servidor J2EE lê o DD e age de acordo com o componente.

A aplicação juntamente com todos os seus módulos é entregue em um arquivo *Enterprise ARchive* (EAR). Um arquivo EAR é um arquivo padrão JAR *Java ARchive* com a extensão .ear. Na versão GUI (*Graphic User Interface*) da ferramenta de desenvolvimento de aplicação J2EE SDK (*Software Development Kit*), é possível criar um arquivo EAR primeiro e adicionar arquivos JAR e WAR ao EAR. Mas se as ferramentas de empacotamento de linha de comando forem usadas, entretanto, podem-se criar arquivos (JARs) e *Web ARchive* (WAR) primeiro e então criar o EAR. O relacionamento entre esses arquivos e o DD é descrita abaixo:

- cada arquivo EJB JAR contém um DD, arquivos *enterprise bean* e arquivos relacionados;
- cada arquivo JAR da aplicação cliente contém um DD, arquivos de classes para aplicação cliente e arquivos relacionados;
- cada arquivo WAR contém um DD, arquivos de componentes web e recursos relacionados.

Usando módulos e arquivos EAR, torna-se possível montar um número de diferentes aplicações J2EE usando somente alguns dos mesmos componentes. Nenhuma codificação extra é necessária, basta montar vários módulos J2EE dentro dos arquivos EAR do J2EE.

4.11 Implementação de Referência

Todas as tecnologias J2EE devem ter um *Compatibility Test Suite* (CTS) associado, que verifica que o produto implementa corretamente o padrão. O J2EE CTS compreende

5.000 programas que testam os detalhes específicos definidos pelas especificações. A Sun Microsystem requer que todas as tecnologias oferecidas sofram os testes correspondentes. Os testes também verificam a interoperabilidade das tecnologias J2EE como especificada pela especificação J2EE [Wor01].

A especificação não é considerada completa até que a tecnologia tenha uma *Reference Implementation* (RI) correspondente, que deve passar pelo CTS [Wor01].

A RI demonstra que é possível implementar a especificação, sem causar problemas de especificação e verifica a corretude do conjunto de teste. O J2EE RI não é um software comercial, e na verdade, não pode ser adquirido comercialmente ou redistribuído [MIC02b].

O J2EE SDK é uma definição operacional não-comercial da plataforma e especificação J2EE disponível gratuitamente pela Sun Microsystems para demonstrações, prototipagem, e uso educacional. Ele vem com o servidor de aplicações J2EE, servidor web, banco de dados relacional, API's J2EE e conjunto completo de ferramentas de desenvolvimento [MIC02b].

4.12 APIs Java para XML

A plataforma J2EE oferece algumas APIs que possibilitam o processamento XML, e consequentemente, a implementação de Web Services. Entre elas estão[ARM03]:

- JAXP (*Java API for XML Processing*) - executa o processamento de documentos XML utilizando vários *parsers* (analisadores gramaticais);
- JAXB (*Java Architecture for XML Binding*) - processa o documento XML usando um esquema derivado das classes dos componentes *JavaBeans*;
- JAX-RPC (*Java API for XML-based RPC*) - envia uma chamada de método SOAP para as partes remotas envolvidas e recebe, posteriormente, os resultados;
- JAXM (*Java API for XML Messaging*) - envia uma mensagem SOAP pela rede de uma maneira padronizada;
- JAXR (*Java API for XML Registries*) - fornece uma padronização para acessar os registros de negócios e compartilhar informações.

Essas APIs podem ser classificadas em duas categorias, as associadas ao processamento de documentos (JAXP e JAXB) e as orientadas a procedimentos (JAX-RPC, JAXM e JAXR), que serão descritas brevemente na próxima seção.

4.13 JAXP (*Java API for XML Processing*)

Essa API facilita o processamento de dados XML usando aplicações em Java [MIC02d].

A JAXP suporta alguns analisadores sintático (*parsers*) padrões como o SAX (*Simple API for XML Parsing*) [ORT00], o DOM (*Document Object Model*) [ORT00] ou o XSLT (*XML Stylesheet Language Transformation*) [ORT00]. Enquanto o SAX executa a análise sintática dos dados como um conjunto de eventos, o DOM constrói uma árvore estruturada para representação dos dados. Já o padrão XSLT permite o controle sobre a apresentação do dados, habilitando a conversão dos dados para outros documentos XML ou para outros formato, como o HTML.

A JAXP fornece o suporte a *namespace*, impedindo o conflito de nomes entre os vários esquemas utilizados pelo documento.

A flexibilidade também é uma característica da JAXP, visto que ela permite a utilização de qualquer *parser* compatível com XML de dentro de uma aplicação. Isso é possível devido a camada de "plugabilidade" fornecida por essa API, que permite que uma implementação das APIs do SAX ou do DOM seja conectada.

O principal pacote da API JAXP é o javax.xml.parser. Dentro do pacote podemos encontrar duas implementações que são representadas pelas classes: *SAXParserFactory* e *DocumentBuilderFactory*. Esta última é a que permite lidar com o documento XML como um objeto no padrão DOM. A Figura 4.6 apresenta o diagrama de componentes da JAXP. A idéia é apresentar as diversas APIs envolvidas e a hierarquia entre as mesmas.

4.14 JAXB (*Java Architecture for XML Binding*)

A API JAXB possibilita a geração classes Java a partir de esquemas XML. Para tanto, ela fornece métodos para transformar um documento XML em uma árvore de conteúdo de objetos Java e vice-versa. Ou seja, a JAXB fornece uma maneira conveniente e rápida de amarrar um esquema XML a uma representação em Java, facilitando a incorporação de dados XML e o processamento de funções em aplicações Java sem conhecer o XML profundamente.

A JAXB oculta os detalhes sobre a relação com o SAX e o DOM, concentrando-se na geração de classes que descrevem apenas os dados definidos nos esquemas fontes. Dada a portabilidade referentes aos dados XML junto com o código portável de Java, é possível criar aplicações e Web Services leves e flexíveis.

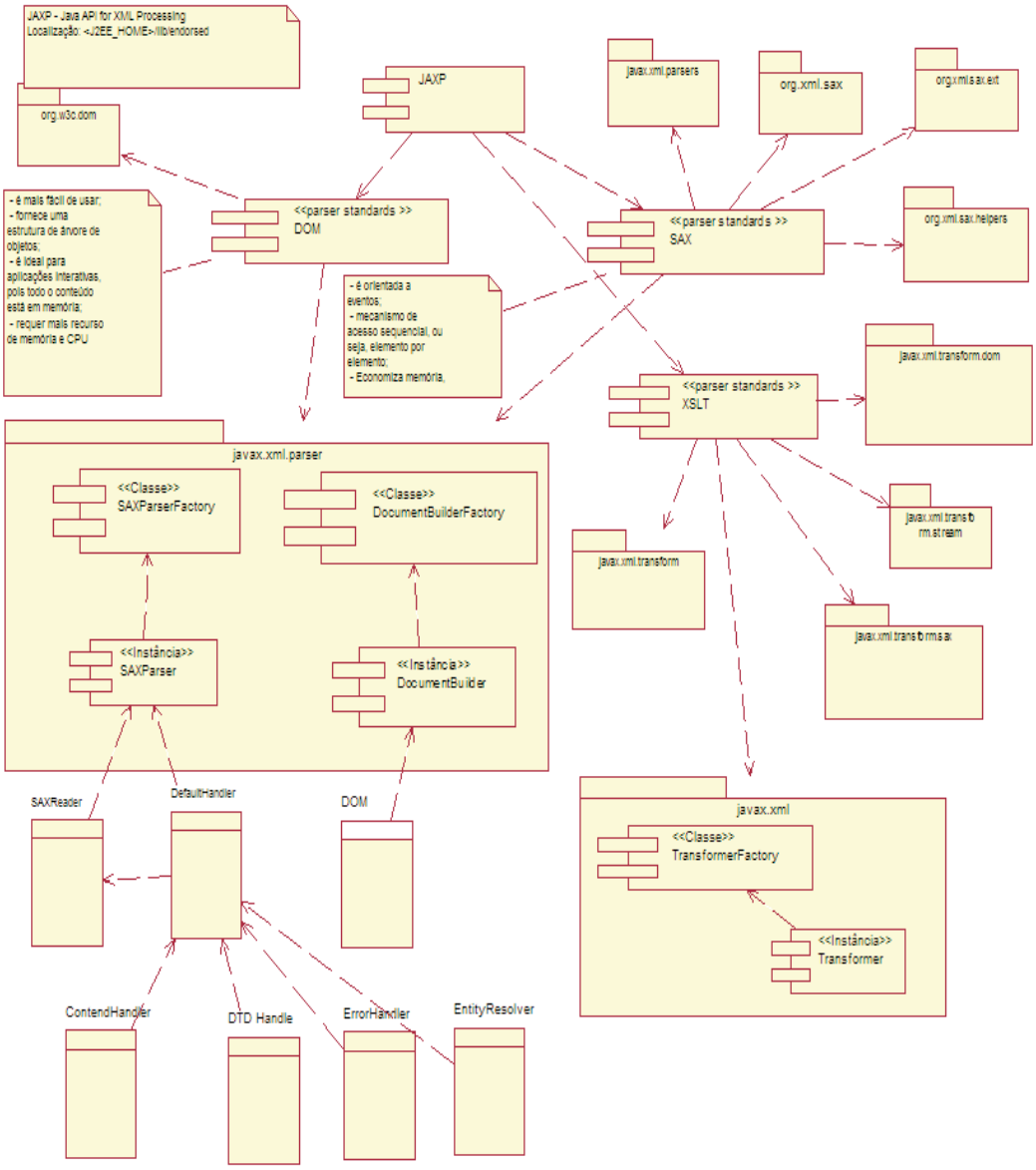

Figura 4.6. *Diagrama de Componentes da JAXP*

4.15 JAX-RPC (*Java API for XML-based RPC*)

Essa API é serve para desenvolver e usar os Web Services, pois é uma coleção de procedimentos que podem ser invocados por um cliente remoto na Internet.

Um Web Service, ou seja, uma aplicação servidora que implementa os procedimentos disponibilizados para os clientes, é publicada em um container do lado do servidor. Esse

container pode ser um container servlet, como o TOMCAT, ou um container Web no servidor J2EE.

Dentre as características mais importante da JAX-RPC estão [ARM03]:

- interoperabilidade - um cliente escrito em uma linguagem diferente de Java pode acessar um Web Service desenvolvido e disponibilizado na plataforma Java. O suporte ao SOAP e ao WSDL promovem a sua interoperabilidade. Como citado no capítulo 4, o SOAP define uma padrão para troca de mensagem baseado em XML. Devido ao suporte ao SOAP, a JAX-RPC é implementada como uma mensagem SOAP pedido/resposta. Além do suporte ao SOAP, essa API também suporta o WSDL, que torna a descrição do do Web Service portável e padronizada, cujo padrão é baseado em XML; e
- facilidade de uso - todas os detalhes complicados envolvidos na tecnologia RPC (*Remote Procedure Call*) são executados automaticamente pela JAX-RPC, facilitando o desenvolvimento de Web Services.

4.16 JAXM (*Java API for XML Messaging*)

Essa API fornece uma maneira padronizada baseada no SOAP (SOAP 1.1) [W3S02] para enviar documentos XML sobre a Internet a partir da plataforma Java.

Geralmente, um fornecedor de mensagem é utilizado por uma empresa. Nesse caso, todas as mensagens JAXM são transportadas através dele. Sua principal responsabilidade é deixar transparente a rota e os detalhes necessários para o transporte de uma mensagem. A escolha da utilização do JAXM possui algumas vantagens em relação ao JAX-RPC, tais como:

- transporte de mensagem em uma direção assíncrona, ou seja, qualquer parte pode se comunicar sem independente da resposta do par integrante da comunicação;
- roteamento de uma mensagem para mais de um destino; e
- transporte de mensagem confiável com características como garantia de entrega;

4.17 JAXR (*Java API for XML Registries*)

A JAXR fornece uma maneira conveniente de acessar registros de negócio de na Internet. Esses registros referem-se às páginas amarelas, descritas no Capítulo 3, que contém uma listagem, dos negócios e dos produtos e serviços oferecidos por eles.

A JAXR provê aos desenvolvedores Java uma maneira padrão para usar os registros que estão baseados em padrões abertos, como o ebXML (*e-business XML*), ou especificações de consórcios industriais, como o UDDI [Sec00].

As empresas podem registrar-se como um registro ou podem descobrir outras empresas com as quais desejam interagir.

A JAXR habilita que as empresas acessem registros padrões de empresas a partir da linguagem Java, aumentando a necessidade da sua utilização para permitir a colaboração através de Web Services.

Como a proposta deste trabalho é fazer uma aplicação-cliente para acessar Web Services já disponíveis, a API JAXR não será utilizada, visto que a mesma deve ser usada para implementar um fornecedor de Web Service.

Neste capítulo foram apresentadas as características gerais sobre a plataforma J2EE e, especificamente, os recursos fornecidos para implementação de Web Service. Foi possível observar que os recursos são complexos e não são de fácil entendimento e utilização. Mas é importante ressaltar que a plataforma oferece recursos que viabilizam o desenvolvimento de Web Services. A forma como esses recursos foram explorados e utilizados neste trabalho será apresentada no próximo capítulo, no qual é mostrada a implementação da aplicação cliente de Web Services.

Capítulo 5

Integração de Sistemas em Ambientes Heterogêneos - Um Estudo de Caso

Neste capítulo será descrito um cenário que possibilita o estudo de caso para integração de sistemas em ambientes heterogêneos. Além da descrição, serão apresentados o escopo, a modelagem da aplicação por meio dos diagramas da UML (*Unified Modeling Language*) [BEZ02] e as estratégias de implementação envolvidas na solução do problema de integração.

5.1 Especificação do Problema

O objetivo do estudo de caso deste trabalho é mostrar que a utilização de Web Service torna possível a integração entre aplicações implementadas nas mais diversas linguagens de programação, provendo uma solução interoperável.

A aplicação de integração será implementada na plataforma J2EE utilizando-se suas APIs Java para processamento XML, pois pretende-se mostrar que os recursos implementados nessa plataforma satisfazem à especificação de Web Services, deixando seu desenvolvimento fácil e transparente.

A aplicação consiste em fornecer uma página Web que disponibilize diversas informações sobre cidades ou países, tais como informações sobre aeroportos, clima, população e localidade. Para tanto, foram acessados alguns Web Services já disponíveis em um repositório (UDDI) público chamado *XMethods* ($http://www.xmethods.com$). Os Web Services utilizados nessa aplicação serão descritos a seguir.

5.1.1 Web Services Acessados

5.1.1.1 Informações Sobre Aeroportos

Utiliza-se o Web Service "*AirportInfoWebservice*" que requer o nome da cidade ou país como entrada e o resultado fornecido consiste em informações como código, nome, País, Abreviação do Pais, GMT Offset, Latitude (em grau, minuto, segundo, N/S), Longitude

(em grau, minuto, segundo, N/S). Esse Web Service está implementado na plataforma MS .NET, estando sua descrição disponível em

$http://www.webservicex.net/airport.asmx?wsdl.$

5.1.1.2 Informações Sobre Clima

Essas informações são acessadas por meio do Web Service "*Global Weather*", que requer o nome de uma cidade/país como entrada e retorna como resultado as condições climáticas para diversas cidades. Esse Web Service também está implementado em MS.NET e sua descrição está em

$http://www.webservicex.net/globalweather.asmx?WSDL.$

Faz-se acesso também a um Web Service para conversão de temperaturas chamado "*Temperature conversion*", a conversão é feita de Farenheit para Celsius e vice-versa. Esse possui a implementação em NuSOAP [ASA03]. Sua descrição está em

$http://mywebservices.free.fr/server/Temperature.wsdl$

5.1.1.3 Informações Sobre População

Essas informações são buscadas com o Web Service "*Country - Population Lookup Service*"que fornece a população de um país. Sua implementação foi feita em Apache SOAP [AXP99] e sua descrição está em

$http://www.cs.uga.edu/\ sent/xmethods/CountryInfoLookup.wsdl.$

5.1.1.4 Informações Sobre Localidade

Para buscar essas informações são acessados alguns Web Services, tais como:

***PlaceFinder*:** Retorna a localização x,y para um local em qualquer parte do mundo, está implementada em GLUE. Sua descrição está em
$http://arcweb.esri.com/services/v2/PlaceFinderSample.wsdl$;

***Proximity*:** Retorna os pontos de interesse na proximidade de uma localidade. Esse Web Service foi implementado em GLUE [GLA02], sua descrição está em
$http://arcweb.esri.com/services/v2/Proximity.wsdl$;

Para mostrar a interação entre as funcionalidades da aplicaàò desenvolvida e os Web Services, será descrito na próxima seção o diagrama de caso de uso.

5.1.2 Diagramas de Caso de Uso

Conceitualmente, o Diagrama de Caso de Uso é uma representação das funcionalidades externamente observáveis do sistema e dos elementos externos ao sistema que interagem com ele [BEZ02].

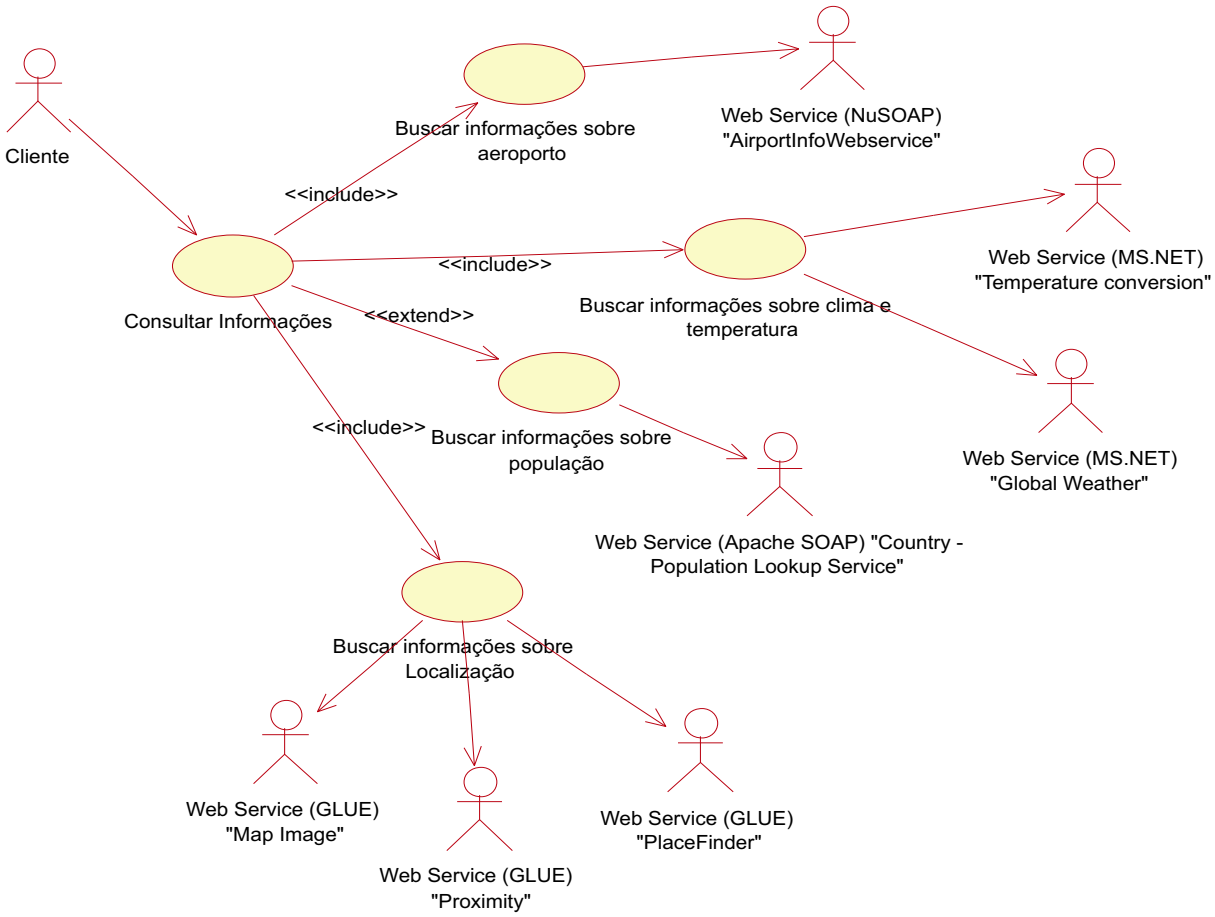

Figura 5.1. *Diagrama de Caso de uso*

No problema exposto, os atores (elementos externos) são os Web Services, disponíveis no *site* XMethods e o cliente da aplicação. Para melhor entendimento do problema, a Figura 5.1 apresenta o seu diagrama de casos de uso. O cliente da aplicação poderá acessar as funcionalidade "Consultar Informações" e, esta por sua vez, acessa outras funcionalidades e cada uma dessas funcionalidades acessa o Web Service adequado.

5.2 Descrição do Ambiente Computacional

Para viabilizar a implementação ou acesso a Web Services, a plataforma J2EE oferece um pacote chamado JWSDP (*Java Web Service Developer Pack*) que reúne as ferramentas necessárias em um único ambiente. Esse ambiente será descrito a seguir.

5.2.1 JWSDP 1.2

O JWSDP 1.2 é um pacote que contém as tecnologias necessárias para simplificar o desenvolvimento de Web Services. As tecnologias disponíveis são [ORT00]:

- XML and Web Services Security;
- JavaServerTM Faces (JSF);
- WS-I Sample Application;
- Java Architecture for XML Binding (JAXB);
- Java API for XML Processing (JAXP);
- Soap with Attachments API for Java (SAAJ);
- Java API for XML Registries (JAXR);
- Java API for XML-based RPC (JAX-RPC);
- JavaServer Pages Standard Tag Library (JSTL);
- Tomcat (Java Servlet and JavaServer Pages container);
- Ant build tool;
- Java WSDP Registry Server;

Na próxima seção serão descritas as ferramentas utilizadas para a efetiva implementação da aplicação.

5.2.2 Descrição das Ferramentas Utilizadas

A aplicação implementada utilizou as seguintes ferramentas disponíveis no JWSDP 1.2:

***Java API for XML Processing* (JAXP) -** como descrito no Capítulo 4, a JAXP manipula parses padrões como o SAX, o DOM e o XSLT. Para implementação da aplicação em questão, foi utilizado o parse DOM (*Document Object Model*);

Wscompile - essa ferramenta é parte integrante das ferramentas JAX-RPC (descrita no Capítulo 4). Gera *stubs*, *ties*, *serializers*, e arquivos WSDL usados nos clientes e servidores JAX-RPC. A ferramenta lê um arquivo de configuração, que especifica um arquivo WSDL, um arquivo modelo ou uma interface de um serviço compilado e armazena os arquivos e classes geradas em um diretório especificado no arquivo de configuração;

Tomcat 5 Servlet/JSP Container - o TomCat consiste na implementação oficial da J2EE para a especificação de Servidor de Aplicação [Apa02a]. A ferramenta de administração TomCat foi utilizada para listar, recarregar, publicar, disponibilizar e remover as aplicações do TomCat. Para a presente aplicação foi utilizado o Java Servlet Container, visto que a aplicação foi baseada em Servlet cujas características foram descritas no Capítulo 4;

Ant build tool - é uma ferramenta de construção baseada em Java que pode ser estendida usando classes java. Os arquivos de configuração são baseados em XML, chamando uma árvore onde várias tarefas são executadas [Apa02b]. Essa ferramenta permite automatizar tarefas comuns de forma simples, diminuindo o tempo de desenvolvimento das aplicações.

Na próxima seção, será descrita a maneira como as ferramentas do JWSDP foram utilizadas para implementação da aplicação proposta.

5.3 Estratégia e Aspectos da Implementação

A primeira atividade realizada para implementação da aplicação foi acessar os documento WSDL dos Web Services para verificar os métodos disponíveis na sua interface. Para tanto, os seguintes passos foram executados:

1) criação do arquivo XML de configuração - esse arquivo deve especificar o endereço do documento WSDL do Web Service, bem como o nome do pacote no qual as classes do Web Service acessado serão armazenadas. É importante salientar que a criação do arquivo de configuração para acesso ao WSDL é característico JWSDP;

2) geração dos STUBS - isso é feito com a ferramenta "wscompile". Para tanto, foi utilizado o comando "wscompile -gen arqconfiguracao.xml". A execução "wscompile -gen"é responsável pela geração dos artefatos (interfaces de *endpoints* do serviço) necessário para implementar uma aplicação cliente, que é o principal objetivo da aplicação descrita neste trabalho.

A seguir é mostrado o arquivo de configuração para acessar o WSDL do Web Service AirportInfoWebservice: 1. $<?xmlversion = "1.0"encoding = "UTF-8"?>$
2. $< configuration$
3.$xmlns = "http://java.sun.com/xml/ns/jax-rpc/ri/config»$

4. $< wsdl\ location = "http://www.webservicex.net/airport.asmx?WSDL"$
5.$packageName = "br.ufpe.mestrado»$
6. $< /wsdl >$
7. $< /configuration >$

A seguir será explicada cada linha do código XML acima:

1) refere-se a um código padrão para criação do arquivo de configuração;

2) inicia a etiqueta configuração, "configuration";

3) *xmlns* - define o protocolo de comunicação a ser utilizado, nesse cado, o protocolo utilizado foi o jax-rpc. Como essa API faz parte do JWSDP, deve-se colocar o endereço da página da sun para que o código seja executado;

4) *wsdl location* - é iniciada a etiqueta *wsdl*, deve-se colocar o endereço do fornecedor do Web Service. Este endereço é adquirido por meio do repositório UDDI disponível. Nesse caso, o repositório utilizado foi o "www.xmethods.com";

5) *packageName* - deve-se criar, na máquina na qual a aplicação está sendo desenvolvida, uma hierarquia de diretórios para armazenar os arquivos e classes devolvidas pelo fornecedor do Web Service, que deve ser separada por ".". No presente trabalho, a hierarquia de diretório, é "br.ufpe.mestrado";

6) é finalizada a etiqueta wsdl;

7) é finalizada a etiqueta de configuração,"configuration".

Para a implementação da aplicação-cliente, os seguintes passos foram executados:

1) criação de Tarefas Ant;

2) parse do arquivo XML;

3) implementação e Execução do Servlet; e

4) construção da página HTML para a entrada de dados do usuário da aplicação.

Esses passos serão detalhados a seguir.

5.3.1 Criação de Tarefas Ant

Para facilitar o desenvolvimento da aplicação, foram automatizadas algumas tarefas, tais como:

- criação dos diretórios: A tarefa "prepare"cria os diretórios *build* e *dist*;

- compilação da classe : A tarefa “compile”depende da tarefa “prepare”;
- compilação do Servlet: É executada a partir da tarefa “compile.servlet”;
- publicação do Servlet no Tomcat: É feita com a tarefa “copy.class”;
- teste da aplicação: A tarefa “run”permite testar a aplicação.

A utilização da tecnologia Ant contribuiu com a simplificação e agilização no processo de compilação, publicação no servlet e o teste da aplicação e, consequentemente, foi alcançada mais produtividade durante o desenvolvimento da aplicação.

5.3.2 Parse do Arquivo XML

Primeiramente, foi criada a classe java que manipula o resultado, um arquivo XML, retornado pelo Web Service. O resultado foi armazenado em uma estrutura de dados chamada *Collection* que permite armazenar diversos valores ao mesmo tempo. A utilização dessa estrutura de dados deve-se ao fato do arquivo XML resultante estar sendo manipulado com o *parse* DOM, que requer que o todo conteúdo esteja em memória.

O parse DOM foi utilizado pelo fato de ser fácil de usar, por fornecer uma estrutura de árvore de objetos, sendo ideal para aplicações interativas (como a aplicação em questão), pois todo o conteúdo está em memória. No entanto, requer mais recurso de memória e CPU. A Figura 5.2 apresenta o diagrama de componentes do DOM. A API DOM também possui outras classes e interfaces envolvidas, a criação do diagrama de componentes não só facilitou o entendimento da API como um todo como também de cada classe associada.

Objetivando um melhor entendimento, é apresentado na Figura 5.3 diagrama de sequência que permite visualizar as atividades gerais necessárias para o acesso a um Web Service. O diagrama de sequência dá ênfase à ordenação temporal de mensagens trocadas pelas classes envolvidas, possibilitando o entendimento dos aspectos dinâmicos de uma aplicação [B+01].

Para cada Web Service acessado, é necessária a criação de uma classe para manipular o resultado, arquivo XML, enviado pelo Web Service.

5.3.3 Implementação do Servlet

Como descrito no Capítulo 4, Servlets são classes da linguagem de programação JAVA que dinamicamente solicitam e constroem respostas.

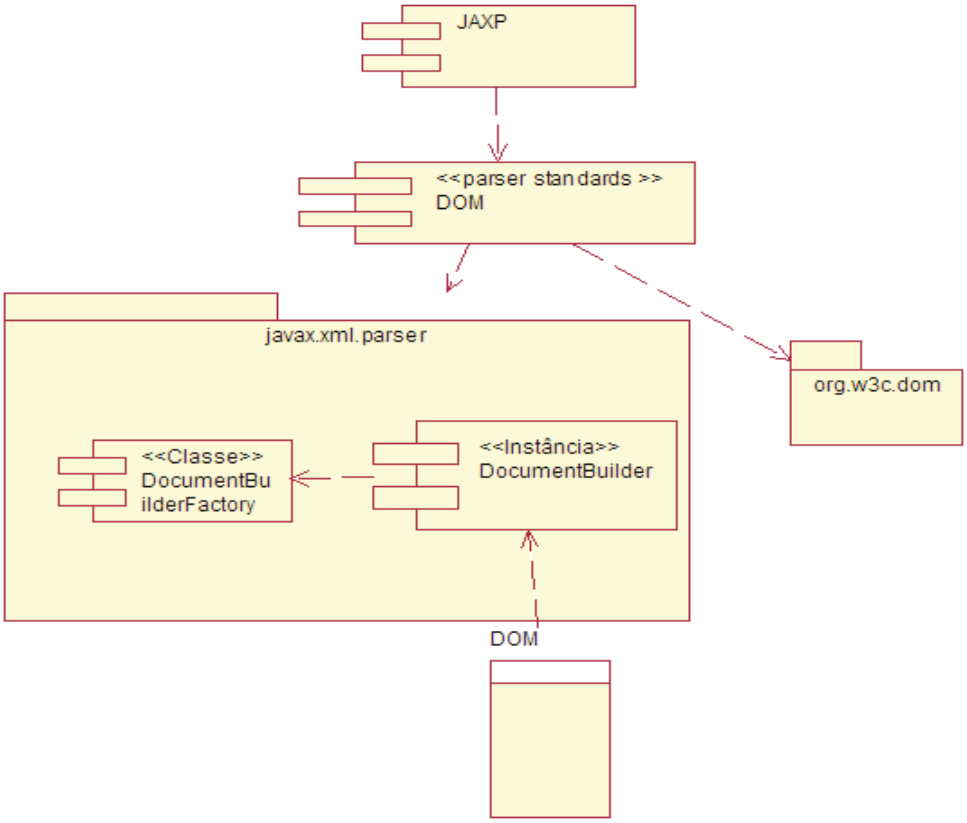

Figura 5.2. *Diagrama de Componentes do DOM*

Essa estratégia de implementação foi escolhida em virtude de oferecer como vantagem o fato possibilitar a utilização de programas Java. Assim, eles permitem a utilização de toda a API Java para a implementação de seus serviços e oferecem adicionalmente portabilidade de plataforma. Além disso, foi considerada a minha experiência com esta tecnologia, bem como o tempo de resposta da utilização de servelt ser melhor do que a utilização de JSP, já que esta última precisa que suas etiquetas sejam primeiramente interpretadas, para que o código seja posteriormente executado.

A API de servlets está concentrada nos pacotes javax.servlet e javax.servlet.http. Classes desses pacotes são utilizadas para desenvolver servlets que serão integrados a servidores.

As quatro principais etapas nessa interação são:

1) cliente envia solicitação ao servidor;

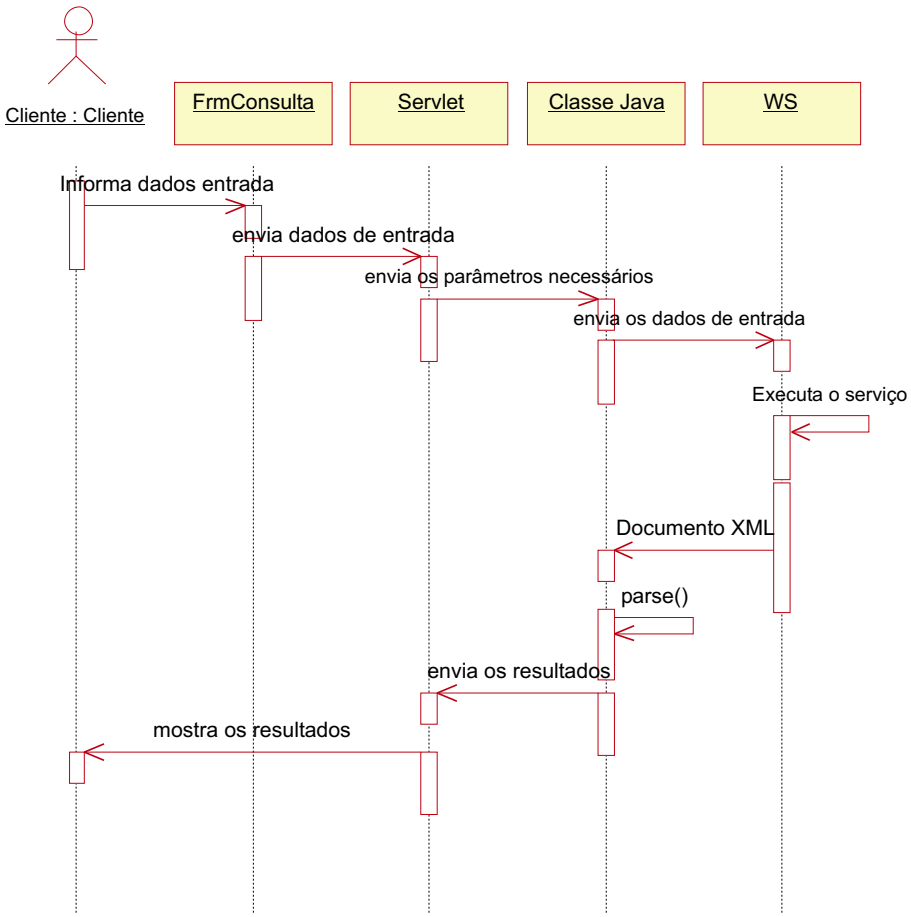

Figura 5.3. *Diagrama de Sequência - Acesso a um Web Service*

2) servidor invoca (através do seu container) o servlet indicado para a execução do serviço solicitado;

3) servlet gera o conteúdo em resposta à solicitação do cliente, eventualmente acessando outros serviços acessíveis através da plataforma Java;

4) servidor repassa o resultado gerado pelo servlet para o cliente como uma resposta HTTP convencional.

Os métodos utilizados na implementação do Servlet são:

- init(): é invocado quando um servlet é carregado pela primeira vez para a máquina virtual Java do servidor;
- destroy(): permite liberar esses recursos (fechar arquivos, escrever o valor final nessa sessão do contador de acessos), sendo invocado quando o servidor estiver concluindo sua atividade; e
- service(): corresponde à resposta a qualquer requisição ao servlet.

Para que um Servlet seja ativado em resposta a uma requisição, é preciso que um container de servlets esteja integrado ao servidor. Neste trabalho, foi utilizado o Web Server TOMCAT por ser um container já disponível no JWSDP, sendo que a publicação do servlet foi feita por meio de uma tarefa automática, "copy.class", na ferramenta Ant.

Figura 5.4. *Página para Entrada de Dados*

Para possibilitar a utilização da aplicação de integração, foi desenvolvida uma página HTML simples, apresentada na Figura 5.4.

A Figura 5.5 apresenta a página resultante da consulta.

5.3.4 Diagrama de Componentes e Implantação

Existem duas maneiras para representar os aspectos físicos de um sistema orientado a objetos, são eles os diagramas de componentes e implantação [B+01].

Os componentes são usados para modelagem de aspectos físicos, como executáveis, bibliotecas, tabelas, arquivos e documentos, que podem residir em um nó. Um compo-

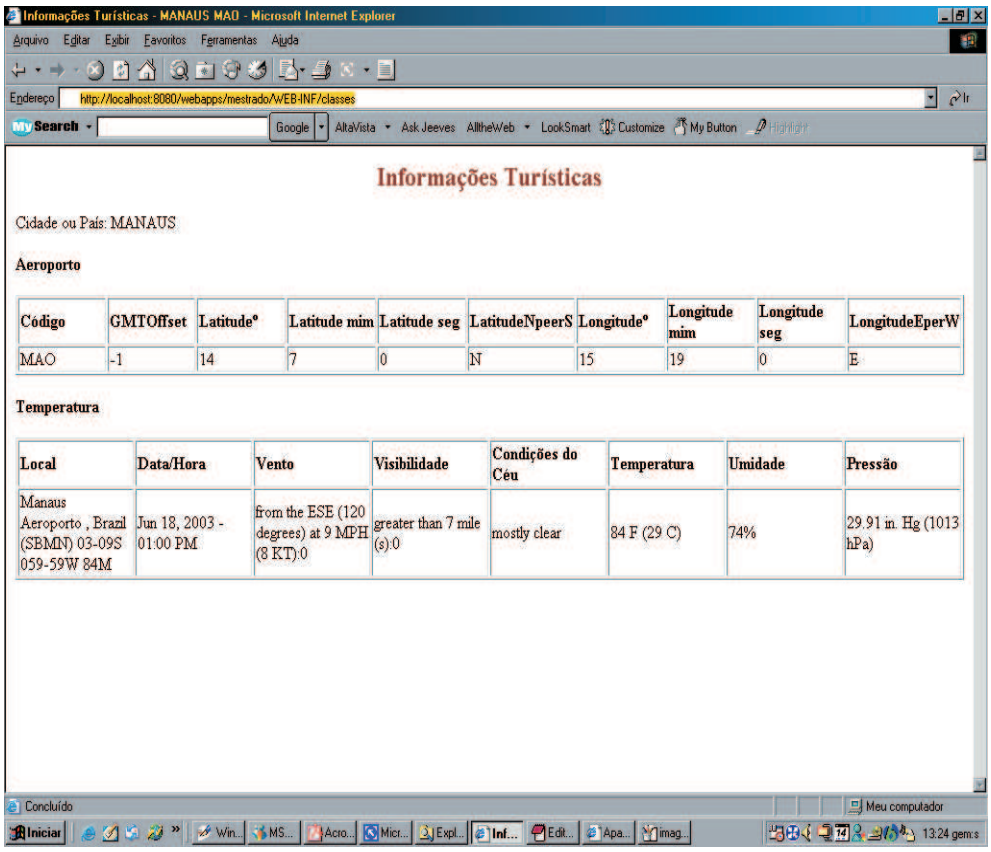

Figura 5.5. *Resultado da Consulta*

nente tipicamente representa o pacote físico de elementos lógicos como classes, interfaces e colaborações [B+01].

O diagrama de componentes da aplicação Turística é ilustrado na Figura 5.6. Neste diagrama é possível visualizar a arquitetura dos componentes integrantes da aplicação que são o servidor de aplicação (Tomcat), a classe ServletQueryWeb.java, que permite que as classe resultantes do acesso aos Web Services do UDDI Xmethods seja acessadas e façam a comunicação com os fornecedores adequados, ao mesmo tempo.

Já o diagrama de implantação mostra a configuração dos nós de processamento em tempo de execução e os componentes que neles existem [B+01]. Esse diagrama é importante para visualizar, especificar e documentar sistemas, sejam eles embutidos, cliente/servidor ou distribuídos. O diagrama de implantação da aplicação desenvolvida é apresentado na Figura 5.7. Neste diagrama é possível visualizar a hierarquia de acesso que a aplicação percorre quando é acessada por um navegador cliente.

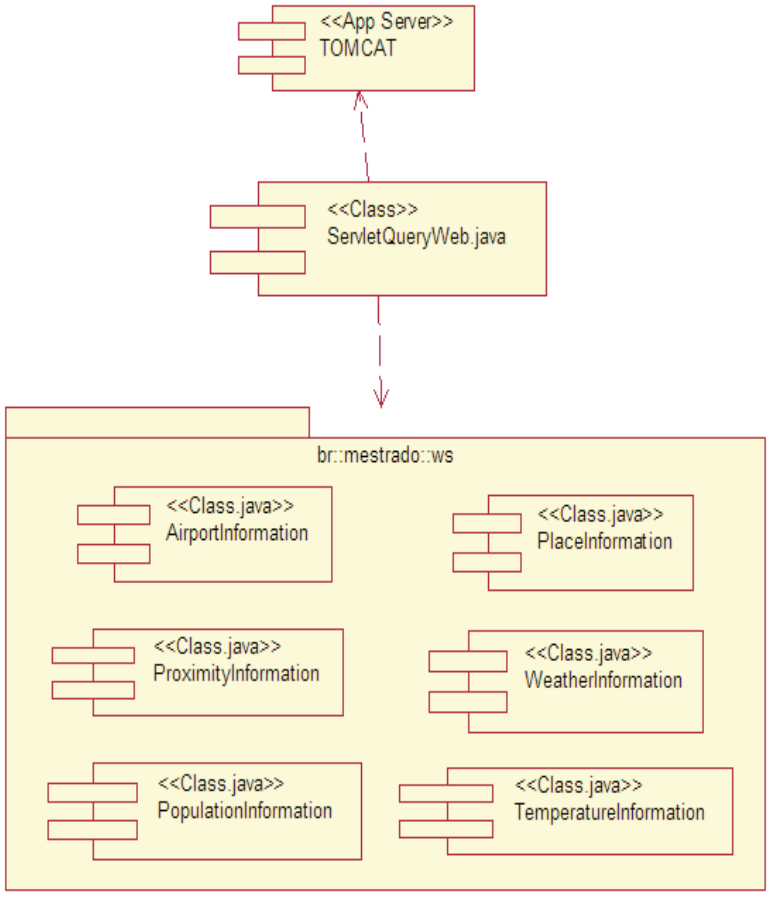

Figura 5.6. *Diagrama de Componentes*

5.4 Resultados Obtidos

A presente aplicação possibilitou comprovar que é possível integrar aplicações heterogêneas utilizando-se Web Services em um ambiente distribuído.

Tal integração deve-se ao fato do Web Service fornecer um padrão para troca de dados, baseado no formato textual (XML) e no protocolo SOAP. Dessa forma, qualquer aplicação, inclusive os sistemas legados, podem interagir entre si a partir de uma aplicação cliente que implemente a comunicação entre os Web Services necessários.

Observou-se a total interoperabilidade entre as aplicações, visto que foi possível integrar informações fornecidas por aplicações implementadas em MS.NET, GLUE (i), NuSOAP (implementação de Web Services em PHP) e Apache, ou seja, linguagens de programação e plataformas distintas e, às vezes até desconhecidas por mim. Mas como a

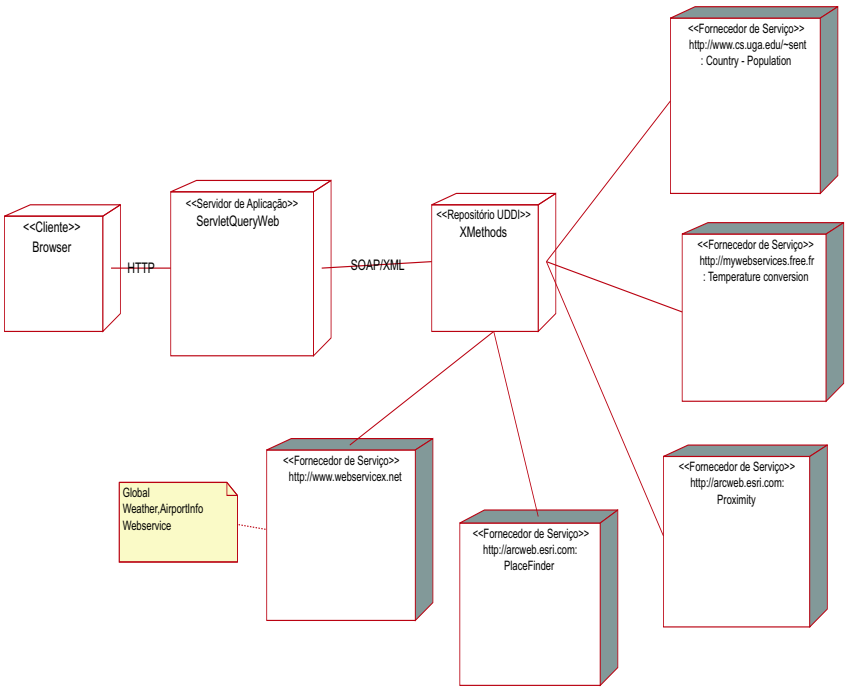

Figura 5.7. *Diagrama de Implantação*

resposta devolvida pelos Web Services eram arquivos XML, bastou utilziar as classe implementadas em Java com a utilização da API DOM para manipular o arquivo recebido.

As ferramentas oferecidas no pacote JWSDP 1.2 facilitaram a implementação da aplicação, visto que o mesmo fornece toda infraestrutura necessária para implementação de Web Services. No entanto, foi necessário conhecer os conceitos referentes programação orientada a objetos, as características da linguagem Java, os conceitos envolvidos na especificação de Web Services, bem como os recursos disponíveis (classe, métodos, APIs e ferramentas) no JWSDP 1.2.

Enfim, pode-se afirmar que a aplicação desenvolvida alcançou o objetivo de forma satisfatória, pois possibilitou a comprovação da real interoperabilidade fornecida pelos Web Services com relação à integração de um ambiente heterogêneo. Para tanto, a aplicação foi testada exautivamente na Web, possibilitando a constatação da efetividade das funcionalidades propostas.

Capítulo 6

Considerações Finais

Neste capítulo é apresentada uma análise sobre as contribuições apresentadas nesta Dissertação, apontando os principais aspectos observados sobre a integração de sistemas em ambientes heterogêneos utilizando Web Services, a plataforma J2EE. Serão descritas também algumas sugestões para trabalhos futuros.

Relevância e Contribuições

Necessidades cada vez maiores no sentido de tornar aplicativos de negócio interoperáveis têm conduzido o foco do desenvolvimento para o modelo da computação distribuída. Neste modelo, em que as camadas se relacionam, um padrão de especificação deve predominar. No quadro atual as especificações EJB(*Enterprise Javabeans*) da Sun Microsystems, COM+ da Microsoft e CORBA são amplamente utilizadas.

Desejar a convergência para um dos padrões seria uma imprudência sob três aspectos. Primeiro, sempre haveria uma parcela de empresas que por motivos diversos, não acompanhariam esta convergência, além de todos sistemas já existentes que não foram desenvolvidos no sistema elite (e não são poucos). A segunda questão, diz respeito aos aspectos técnicos, já que cada especificação das três citadas carregam em alguns casos uma forte ou parcial dependência do ambiente de sistema operacional a qual está associada. O terceiro e último aspecto é, por si só, suficiente para evitar a determinação em torno de uma única especificação, ou seja, a aniquilação da concorrência de fornecedores de tecnologia.

É nesse cenário, que os Web Services surgiram para por ordem ao caos e impulsionar o desenvolvimento de aplicativos interoperáveis. Seu grande mérito é realmente estar desprendido de um fabricante ou grupo, pois se trata de um padrão de desenvolvimento para comunicação entre sistemas. Eles têm sido apresentados como solução a ser adotada pelo canal Internet, mas pode ser utilizado, inclusive, para aplicativos de redes corporativas interligados entre si.

Além disso, o mundo é heterogêneo, por isso, investir em tecnologia de Web Services com base em padrões abertos é a melhor alternativa, em vez de um único sistema operacional e proprietário que o conecte e não se adapte às suas necessidades.

A elaboração deste trabalho, incluindo as etapas de pesquisa e estudos juntamente com a implementação da aplicação de integração, possibilitou a validação e comprovação da interoperabilidade prometida pelos Web Services para integração de sistemas. Já que conseguiu-se integrar os dados fornecidos por aplicações sem a preocupação com a plataforma original das mesmas. A validação e comprovação foi feita através de teste exautivos na aplicação.

Foi observado que o pacote da Sun Microsystems JWSDP 1.2 é uma boa alternativa para o desenvolvimento de aplicações que lidam com Web Service (sejam clientes ou provedoras de serviços). Isso deve-se ao fato desse pacote estar baseado na plataforma J2EE que fornece APIs para processamento XML, pois não faz-se necessário a preocupação com a implementação da infraestrutura básica, tais como segurança e tráfego de informações, podendo-se focar especificamente na implementação da aplicação desejada. No entanto, é de fundamental importância possuir conhecimentos sólidos sobre alguns aspectos, tais como a especificação de Web Services, programação orientada a objetos, bem como mecanismos de publicação dos *servlets*, as próprias APIs e as ferramentas disponíveis no pacote JWSDP 1.2 para que a implementação de aplicações seja possível, visto que os passos para implementação de tais aplicações não são triviais.

Dentre as vantagens observadas com a utilização de Web Services, pôde-se destacar as seguintes: independência da linguagem de programação por se tratar de uma especificação de padrão, facilidades de manutenção, pois a partir de uma interface pré-estabelecida é possível modificar seu funcionamento sem impacto nos clientes da aplicação, facilidade de uso em ambiente com *firewall* e *proxies* devido à utilização do protocolo SOAP, que é baseado em XML e HTTP.

Quanto às desvantagens, observou-se a dependência das aplicações-clientes em relação ao resultado enviado pelo Web Service, pois se houver mudança no mesmo, a aplicação cliente deve ser modificada também. Isto porque as classes implementadas estão preparadas para lidar com um arquivo XML previamente definido. Tal dependência aumenta o custo de manutenção das aplicações-cliente.

Uma das grandes dificuldades durante o desenvolvimento da aplicação foi referente ao acesso aos Web Services. Pois foi decidido utilizar um repositório UDDI gratuito, pela facilidade de acesso, o XMethods, houve problemas para acessar alguns Web Services, pois os mesmos nem sempre estavam disponíveis, dificultando a implementação e os testes. É importante salientar que o escopo da aplicação foi modificado algumas vezes devido esse problemas de acesso.

Outra dificuldade refere-se ao entendimento das ferramentas e dos passos necessários para implementar a aplicação que utilizasse Web Services a partir do JWSPD 1.2, devido ao pouco conhecimento inicial sobre os conceitos envolvidos.

Todos os objetivos propostos no trabalho foram alcançados de forma satisfatória, constatando-se que Web Service é um padrão não proprietário, uma aposta de todos em um formato não binário que, por conseqüência, fornece interoperabilidade real, possibilitando o processamento distribuído e a integração de aplicações em qualquer plataforma, ou seja, em ambiente heterogêneo. Foi observado e constatado que o sucesso dos Web Services deve-se à utilização de XML para troca de dados através do protocolo HTTP, pois, dessa forma, superam as limitações das demais soluções propostas para essa finalidade.

Foi possível constatar que os Web Services existem porque a interoperabilidade não só é possível, mas é uma realidade atual. A interoperabilidade se opera através de *middleware*, um software que permite aos aplicativos funcionarem e se comunicarem com múltiplos sistemas operacionais. O uso de padrões abertos por mais empresas indica que este *middleware* permitirá que sistemas interatuem perfeitamente, sem importar qual sistema operacional ou aplicativo é utilizado. Os padrões abertos permitem que se escolha a infra-estrutura tecnológica, conforme seja mais apropriada e economicamente efetiva para as corporações.

Como proposta para trabalho futuro, sugere-se a utilização de outras plataformas de desenvolvimento (como a MS.NET) para testar os recursos oferecidos para desenvolvimento de Web Service com a finalidade de fazer uma análise comparativa entre tais plataforma e a plataforma J2EE.

Como esta dissertação se concentrou no desenvolvimento de uma aplicação-cliente na plataforma J2EE, outra sugestão refere-se ao desenvolvimento de uma aplicação fornecedora de Web Services para verificar a utilização dos demais recursos do pacote JWSDP, tais como JAXR, JAXM, entre outros.

Referências Bibliográficas

[A+02] Daniel AUSTIN et al. Web Services Architecture Requirements. W3C World Wide Web Consortium, Novembro 2002. http://www.w3.org/TR/2002/WD-wsareqs20021114. Acesso em: 01/03/03.

[Apa02a] Apache Software Foundation. The Jakarta Project:Apache TomCat, 2002. http://jakarta.apache.org/tomcat/. Acesso em:15/06/03.

[Apa02b] Apache Software Foundation. The Apache Ant Project, 2002. http://ant.apache.org/. Acesso em:15/06/03.

[ARM03] Eric ARMSTRONG. The Java Web Services Tutorial. SUN MICROSYSTEM, 2003. http://java.sun.com/webservices/docs/tutorial/doc/index.html. Acesso em:25/02/03.

[ASA03] Ahm ASADUZZMAN. Building XML Web Services with PHP NuSOAP, 2003. http://www.devarticles.com/art/1/414/2. Acesso em: 10/07/03.

[AXP99] The Apache XML Project. Apache SOAP, 1999. http://ws.apache.org/soap/. Acesso em: 30/03/03.

[B+00] Tim BRAY et al. Extensible Markup Language (XML) 1.0. W3C Recommendation, 2000. http://www.w3.org/TR/REC-xml. Acesso em: 05/01/03.

[B+01] Grady BOOCH et al. *Special Edition Using SOAP*. Que, 2001.

[BAN02] Sonai BANSAL. The Web at your (machines) service, 2002. http://www.javaworld.com/javaworld/jw092001/jw0928smsservicep.html. Acesso em: 30/11/03.

[BAR] Julio Cesar BARBOSA. Criptografia de chave pública baseada em curvas elípticas. Master's thesis, COPPE.

[BEZ02] Eduardo BEZERRA. *Princípios de Análise e Projeto de Sistemas com UML*. Editora Campus, 2002.

[BHA00] Mandar BHAGWAT. *Analysis of Simple Object Access Protocol (SOAP)*. Advanced Technology Group Digital Equipament Ltda, 2000.

[BOX00] Don BOX. Simple Object Access Protocol. W3C - World Wide Web Consortium, 2000. http://www.w3.org/TR/SOAP. Acesso em: 15/02/03.

[C+01] Rick CATTELL et al. *J2EE - Criando aplicações comerciais com a plataforma Java 2, Enterprise Edition.* Editora Campus, Rio de Janeiro, 2001.

[C+02] David CHAPPELL et al. *Java Web Services.* O'Reilly, 2002.

[C+03] Roberto CHINNICI et al. Web Services Description Language (WSDL) Version 1.2. W3C Working Draft, 2003. http://www.w3.org/TR/wsdl12/. Acesso em: 20/03/03.

[CHA02] Michael CHAMPION. Web Services Architecture Requirements. W3C World Wide Web Consortium, Novembro 2002. http://www.w3.org/TR/2002/WD-ws-arch-20021114. Acesso em: 15/02/03.

[COH01] Frank COHEN. Myths and misunderstandings surrounding SOAP. *IBM*, 2001.

[COY00] Frank P. COYLE. *XML, Web Services, and the Data Revolution.* Addison Wesley, 2000.

[D+01] H. M. DEITEL et al. *Java: Como Programar.* Bookman, Porto Alegre, 3 edition, 2001.

[FLA99] David FLANAGAN. Java in a nutshel, 1999. http://www.oreilly.com. Acesso em: 25/11/02.

[G+97] Vanderburg GLENN et al. *Maximum Java 1.1.* SAMS NET, 1997.

[GLA02] Graham GLASS. Glue web services platform 2.1, 2002. http://www.webservices.org/index.php/article/articleview/14/1/9/. Acesso em: 15/07/03.

[IBM02] IBM. Publishing your services:UDDI. Tutorial, 2002. https://www6.software.ibm.com/developerworks/education/ws-psuddi/index.html. Acesso em: 20/03/03.

[INM97] William H. INMON. *Como Construir O Data Warehouse. Trad. 2a. Edição.* Ana M. Netto Guz, Rio de Janeiro, 1997.

[K+02] Casey KOCHMER et al. *Integrating XML and Web Services in Your JSP Application.* Pearson Education, 2002.

[KAO02] James KAO. Developer's guide to building XML-based Web Services, 2002. http://www.theserverside.com/resources/articles/WebServices-Dev-Guide/article.html. Acesso em: 05/02/03.

[L+96] R. LINS et al. *Garbage Collection.* John Wiley and Sons, 1996.

[LAU01] Christophe LAUER. Introducing Microsoft dotNET, 2001. http://www.freevbcode.com/ShowCode.Asp?ID=2171. Acesso em: 15/03/02.

[LIN99] David S. LINTHICUM. *Enterprise Application Integration*. Addison Wesley, 1999.

[LIN01] David S. LINTHICUM. Next-generation EAI: Eight prophecies for 2001, 2001. http://eai.ebizq.net/str/linthicum_1.html. Acesso em: 10/01/03.

[MIC98] SUN MICROSYSTEM. The Java Language: An Overview, 1998. http://java.sun.com/docs/overviews/java/java-overview-1.html. Acesso em: 07/05/03.

[MIC99] SUN MICROSYSTEM. Java Remote Method Invocation, 1999. http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiobjmodel2.html. Acesso em:15/04/03.

[MIC02a] SUN MICROSYSTEM. A Plataforma J2EE. http://www.sun.com.br/service/consultoria/java/java2.html, 2002.

[MIC02b] SUN MICROSYSTEM. The J2EE Tutorial, 2002. http://java.sun.com/j2ee/tutorial/. Acesso em: 06/07/03.

[MIC02c] SUN MICROSYSTEM. Java RMI over IIOP. http://java.sun.com/products/rmi-iiop/, 2002.

[MIC02d] SUN MICROSYSTEM. Web Services Made Easier The Java APIs and Architectures for XML, 2002. http://java.sun.com/xml/webservices.pdf. Acesso em: 03/02/03.

[MUE01] John Paul MUELLER. *Special Edition Using SOAP*. Que, 2001.

[NAK01] Shin NAKAJIMA. A SOAP-based Infrastructure for Service Broker. *Japan: Networking Research Laboratories, NEC Corporation*, 2001.

[OMG03] OMG, Object Management Group. INTRODUCTION TO OMG'S SPECIFICATIONS:CORBA, 2003. http://www.omg.org/gettingstarted/specintro.htm#CORBA. Acesso em: 10/05/03.

[ORT00] Ed. ORT. Java Web Services Developer Pack, 2000. http://developer.java.sun.com/developer/technicalArticles/ WebServices/WSPack/. Acesso em: 13/12/03.

[PAC00] Viviane PACHECO. EAI: a sigla da integração dos sistemas internos e externos, 2000. http://www.neogrid.com.br/portugue/imprensa/imprensa/surftrade/25.../SurftradeBrasil.html. Acesso: 05/01/03.

[R+01a] Ed. ROMAN et al. J2ee vs. microsoft.net: A comparison of buiding xml-based web services, 2001. http://www.theserverside.com/resources/articles/J2EE-vs-DOTNET/article.html. Acesso em: 12/12/02.

[R+01b] William A. RUH et al. *Enterprise Application Integration: A Wiley tech brief*. John Wiley & Sons, 2001.

[RAJ00] Gopalan Suresh RAJ. A detailed comparison of CORBA, DCOM and Java/RMI, 2000. http://my.execpc.com/ gopalan/misc/compare.html. Acesso em: 16/05/03.

[RIC01] Ivan Luiz Marques RICARTE. Programação orientada a objetos: Uma abordagem com java, 2001. http://www.dca.fee.unicamp.br/courses/PooJava/progdist.html. Acesso em: 12/02/03.

[ROM01] Ed ROMAN. *Mastering Enterprise JavaBeans and the Java 2 Plataform, Enterprise Edition*. Wiley Computer Publishing, Canada, 2001.

[S+01] Odd Are SAEHLE et al. SOAP and Web Service, 2001. http://www.mas.com.br/Artigos/Soap_Web.htm. Acesso em: 14/03/03.

[S+03] Robert C. SEACORD et al. *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*. Addison Wesley, 2003.

[Sec00] OASIS Member Section. UDDI Technical White Paper. OASIS Member Section - UDDI.ORG, 2000. http://www.uddi.org/. Acesso: 12/04/03.

[Soc01] IEEE Computer Society. .net explained. *Computer Innovative Technology for Computer professionals*, August 2001.

[SOM95] Ian SOMMERVILLE. *Engenharia de Software*. Addison Wesley, 5a. edition, 1995.

[SUO02] Zdenek SUOBODA. Creating a web service in 30 minutes, 2002. http://www.theserverside.cm/resources/articles/Systinet-web-services-part-1/ article.html. Acesso em: 15/04/03.

[T+03] Scott TILLEY et al. Approaches to Legacy System Evolution. 2003.

[TEC02] MICROSOFT TECHNOLOGIES. Introduction to SOAP, 2002. http://www.stylusinc.net/technology/soap/soap1.shtml. Acesso em: 02/03/03.

[W+02] Jacob WEINTRAUB et al. Introduction to Web Service and WSDK. *IBM*, 2002. Tutorial.

[W3S02] W3Schools. Introduction to SOAP, 2002. http://www.w3schools.com/soap/soap_intro.asp. Acesso em: 19/04/03.

[W3S03a] W3Schools. DTD tutorial, 2003. http://www.w3schools.com/dtd/default.asp. Acesso em: 18/04/03.

[W3S03b] W3Schools. XML Schema Tutorial, 2003. http://www.w3schools.com/schema/default.asp. Acesso em: 19/01/03.

[Wor01] Java World. A walking tour of J2EE: What makes the J2EE platform, 2001. http://www.javaworld.com/javaworld/jw072001/jw0727enterprisejava.html?. Acesso em: 20/11/02.

[Z+00] Elizabeth D. ZWICKY et al. *Building Internet Firewalls*. O'Reilly, 2nd edition edition, 2000.