UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS–GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Maidi Terezinha Dalri

INTEROPERABILIDADE E REUSABILIDADE EM SERVIÇOS WEB ATRAVÉS DA INTEGRAÇÃO ENTRE ORIENTAÇÃO A OBJETOS E A ARQUITETURA ORIENTADA A SERVIÇOS.

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação.

Prof. Mauro Rosienberg

Florianópolis, junho 2003

Folha das assintauras

"A vitória existe para aqueles que estão sempre dispostos a recomeçar."
Anônimo

"O futuro pertence àqueles que acreditam na beleza de seus sonhos."
Eleanor Roosevelt

"A inteligência é uma construção do sujeito que enriquece os objetos externos"
Jean Piaget

Ao meu eterno companheiro e amigo Carlos,

com todo meu amor e carinho!

AGRADECIMENTOS

Desde o início das atividades que fizeram parte da realização deste Curso de Mestrado em Ciência da Computação até a conclusão do mesmo, muitas coisas importantes aconteceram em minha vida, que se refletiram num aprendizado profundo e num amadurecimento pessoal e profissional extremamente significante. Nessa caminhada, muitas pessoas tiveram uma parcela significativa de participação e envolvimento. Não é fácil encontrar palavras para expressar a alegria e a felicidade em troca do auxílio recebido, mas gostaria de registrar um agradecimento especial as seguintes pessoas e instituições:

Aos meus colegas e amigos do departamento de informática, pela dedicação, carinho e profissionalismo com o qual sempre assumiram suas atividades e pelo apoio recebido no decorrer do curso, tanto na realização de pesquisas quanto no abraço fraterno nas horas de angústia.

Aos colegas professores pela paciência, compreensão e auxílio nas horas em que precisei me ausentar das atividades docentes e de coordenação.

À Sociedade Educacional Três de Maio, na pessoa do Sr. Seno Leonhardt, diretor geral e da Sra. Zenaide Tesche Heimerdinger, vice–diretora, pelo apoio, carinho, confiança e incentivos dispensados a minha pessoa no decorrer destes dois anos de intensas atividades.

À Universidade Federal de Santa Catarina na pessoa do professor Roberto Willrich, pela oportunidade de realizarmos um Curso de Pós–Graduação em nível de Mestrado em Ciência da Computação, bem como pelo acompanhamento e dedicação nas atividades realizadas no decorrer do curso.

Aos mestres que integraram o quadro docente deste curso e, que souberam de forma extremamente profissional nos auxiliar no processo de construção e produção dos nossos próprios saberes dentro dessa importante área da tecnologia.

Ao meu querido orientador, professor Mauro Roisenberg, pelo seu profissionalismo, exigência e apoio, que se constitui num pilar de extrema importância para o êxito na conclusão do presente trabalho.

Aos meus familiares e amigos, pelo apoio e incentivo recebidos.

De forma muito especial ao meu filho Carlos Eduardo Ceccon e ao meu eterno companheiro Carlos Alberto Ceccon, pelo seu carinho, amor, paciência e auxílio nessa importante caminhada, bem como pela compreensão nas minhas horas de ausência e angústia. Com certeza vocês foram os sustentáculos para a concretização dessa conquista.

Por fim não poderia deixar de registrar um agradecimento especial Àquele que nos dá coragem, força e entusiasmo para perseguirmos nossos objetivos: Deus. Um agradecimento especial a Deus, por possibilitar as escolhas certas e guiar os nossos caminhos, nos conduzindo sempre a um rumo certo.

LISTA DE ABREVIATURAS

ALGOL – Algorithmic Oriented Language
ANS – Advanced Network and Services
ANSP – Academic Network at São Paulo
API – Application Programming Interface
ARPA – Advanced Research and Projects Agency
B2B – Business–to–Business
BI – Business Intelligence
BML – Bean Markup Language
BNF – Backus–Naur Form
BSML – Bioinformatic Sequence Markup Language
BTP – Bussines Transaction Protocol
CBL – Common Business Library.
CDF – Channel Definition Format
CML – Chemical Markup Language
COL – Component Object Library
COM – Component object model
CORBA – Commom Object Request Broker Architetura
CPAs – Collaboration Protocol Agreements
CPML – Call Policy Markup Language
CRM – Customer Relationship Mangement
CSS – Cascading Style Sheets
cXML – CommerceXML
DCE – Distribution Computing Environment
DCOM – Distributed Component Object Model
DISA – Data Interchange Standards Association
DOM – Document Object Model
DRI – Defense Research Internet
DTD – Document Type Definition
EACML – Extensible Access Control Markup Language

EAI – Enterprise Application Interchange
EBNF – Extended Backus–Naur Form
EBXML – Eletronic Business eXtended Markup Language
EDI – Electronic Data Interchange.
FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo
GIOP – General Inter–ORB Protocol
GML – General Markup Language
GUID's – Globally Unique Identifiers
HL7 – Health Level 7
HTTP - Hypertext Transfer Protocol
HTTP–R – HTTP Reliable
IAB – Internet Architeture Board
IDL – Interface Definition Language
IETF –Internet Engineering Task Force
IFX – Interactive Financial Exchange.
IIOP – Internet Inter–OP
IML – Instrument Markup Language
ISE – Eiffel Software Inc.
ISO – International Organization for Standardization
ISOC – Internet Society
ISP – Internet service provider
JASS – Java Authentication and Authorization Service
JCE – Java Cryptography Extension
JDBC – Java DataBase Connectivity
JMS – Java Messaging Server
JNI – Java Native Interface
JSSE – Java Secure Socket Extension
JVM – Java Virtual Machine
LNCC – Laboratório Nacional de Computação Científica
MAC – Message Authentication Code
MOF – Meta Object Facility
MPW – Macintosh Programmer's Workshop
NFS – National Science Foudation
OASIS – Organization for the Advancement of Structured Information Standards
OFE – OFX Open Financial Exchangeou OFX
OLE – Object Linking and Embebding
OMG – Object Management Group

OO – Object Oriented
ORB – Object Request Broker
OSD – Open Software Description
QoS – Quality ofService
RMI – Remote Method Invocation
RNP – Rede Nacional de Pesquisa
RPC – Remote Procedure Call
SAML – Security Access Markup Language
SAX – Simple API for XML
SDK – Software Developer Kit
SGF – Structured Graf Forma,
SGML – Standard Generalized Markup Language
SLA – Service Level Agreement
SMTP – Simple Mail Transfer Protocol
SOA – Services Oriented Architeture
SOAP – Simples Object Access Protocol
SS7– Signaling System 7.
SSL – Secure Sockets Layer
SSTC – Security Services Technical Committee
TCP – Transfer Control Protcol
TLS – Transport Layer Security
UDDI – Universal Description, Discovery and Integration
UFRJ – Universidade Federal do Rio de Janeiro
UML – Unified Modeling Language
URI – Uniform Resource Identifier
VPNs – Virtual Private Networks
W3C – World Wide Web Consortium
WSDL – Web Services Description Language
WSEL – Web Services EndPoint Language
WS–I – Web Services Interoperability Organization
WSIF – Web Services Invocation Framework
WSUI – Working Group web Services User Interface
XFDL – Extensible Forms Description Language
X–KISS – Key Information Service Specification
XKMS – XML Key Management Services
X–KRSS – XML Key Registration Service
XMI – XML Metadata Interchange

XML – eXtensible Markup Language

XSL – Extensible Style Language.

XSLT – XSL Transformation.

X–TASS – XML Trust Assertion Service

XUL – XML–based User Interface Language

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

O mundo tecnológico evoluiu consideravelmente nas últimas décadas. Inúmeras iniciativas em termos de ferramentas, sistemas operacionais, aplicações, linguagem de programação, e demais soluções computacionais foram concebidas por universidades, instituições, organizações e pesquisadores da área de tecnologia. No entanto, como não existia nenhuma padronização as tecnologias desenvolvidas tomaram, na sua grande maioria, rumos diferentes e se chegou a um ponto em que se pode constatar uma enorme dificuldade no relacionamento entre aplicações distintas, que se sobrepõe inclusive à relação homem–máquina.

Essa realidade levou a área de tecnologia a buscar ao longo dos últimos anos soluções para a integração de plataformas, sistemas e aplicações, alcançando avanços significativos que, na sua evolução culminam hoje nas tecnologias oferecidas pelos *serviços web*, mostrando a real possibilidade da interoperabilidade entre aplicações, linguagens e plataformas operacionais distintas.

Neste sentido o presente trabalho tem como objetivo principal mostrar que a integração entre o paradigma de orientação a objetos e a arquitetura orientada a serviços (*serviços web*) é uma das principais formas de garantir a interoperabilidade e a reusabilidade de *softwares*, principalmente nas aplicações de Internet, pois estas tecnologias encontram-se embasadas em padrões abertos e altamente flexíveis utilizando-se principalmente de recursos como XML (eXtensible Markup Language).

ABSTRACT

The technological world has considerably evolved in the last decades. Several initiatives concerning tools, operating systems, applications, programming languages e other computing solutions have been conceived by universities, institutions, organizations, and researchers in the technology area. However, there was no pattern and most developed technologies took different ways and then it was noticed a huge difficulty among distinct application relationships. And these are even above the man–machine relation.

This reality led technology area to search for solutions during the years to platform integration, systems and applications, reaching meaningful advances which nowadays are the technologies offered by web services, showing the real possibility of interoperability among applications, languages and distinct operating platforms.

Thus, this study main goal is to show that the integration between the object orientation paradigm and the service oriented architecture (webs services) is one of the main ways to guarantee the interoperability and the software reusability especially on Internet applications, because these technologies are based on open standards and quite flexible using resources like XML (Extensible Markup Language).

1 INTRODUÇÃO

Atualmente dispõe–se de várias ferramentas que possibilitam a uma empresa integrar e garimpar informações em sua base de dados.

O mundo tem a sua disposição a maior rede de computadores jamais imaginada: a Internet. Porém, a maioria das instituições ainda não utiliza os recursos e possibilidades oferecidas no objetivo de construir meio ágil, seguro, eficiente, prático e, acima de tudo, interativo, de se trabalhar com essa grande massa de conhecimento dentro das atividades diárias.

Os recursos não são utilizados de forma otimizada e poucas instituições sabem que é possível realizar a troca de informações entre instituições parceiras de forma dinâmica, com gerenciamento das próprias aplicações.

Na verdade, um dos maiores problemas que contribuiu para esta realidade era a barreira da interoperabilidade de plataformas, sistemas operacionais e bases de dados.

1.1 Motivação e Justificativa

No decorrer da evolução das tecnologias de rede, várias tentativas de interoperabilidade entre diferentes plataformas foram desenvolvidas e testadas. Algumas destas ações acabaram concretizando–se em produtos que prometiam oferecer a solução para o problema, como por exemplo: CORBA, DCOM e RMI. Entretanto essas soluções apesar de melhorarem em muito os problemas existentes, acabavam esbarrando em *proxys*, ou outras particularidades dos sistemas, uma vez que não se constituíram em padrões totalmente abertos, ou seja, CORBA, DCOM e RMI constituem–se em *middlewares* que apesar da proposta da interoperabilidade continuam atrelados de alguma forma ou a plataformas ou a linguagens de programação.

Na evolução desses *middlewares* surge uma nova arquitetura de *software*: a arquitetura orientada a serviços, que dá base aos serviços web (*web services)* e todas as tecnologias envolvidas, como por exemplo, SOAP (*Simples Object Access Protocol*), UDDI (*Universal Description, Discovery and Integration*) e WSDL (*Web Services Description Language*).

As soluções apresentadas por essas tecnologias para a resolução do problema de interoperabilidade apresentam como principais pilares os recursos oferecidos pelo paradigma da orientação a objetos e a interoperabilidade possibilitada pelo uso da XML (*eXtensible Markup Language*).

Um serviço web é uma aplicação que aceita solicitações de outros sistemas através da internet, oferecendo funcionalidades específicas. A proposta dos serviços web é a de que as aplicações possam procurar informações sem a intervenção de pessoas e interagir uma com as outras.

Um aspecto extremamente positivo que pode ser citado no uso destas novas soluções reside na facilidade de manutenção das interfaces colocadas à disposição dos usuários do sistema e a independência destes em relação ao local ou estação de trabalho a ser utilizada.

As tecnologias envolvidas nos serviços web: SOAP, WSDL e UDDI estão embasadas numa linguagem de marcação aberta, totalmente interoperável: a XML. O protocolo SOAP possibilita a execução de métodos com passagem e retorno de parâmetros através de um arquivo XML que trafega pela *web* via HTTP. O WSDL é a definição da interface de um serviço web e a UDDI é responsável por efetuar os registros dos serviços web.

Essas tecnologias são novas e não possuem muitas publicações que versam sobre o assunto e, muito menos que as interelacionem com o paradigma da orientação a objetos. Um estudo que possa verificar a relação existente entre ambos com certeza trará uma maior compreensão em si e possibilitará a socialização do conhecimento acerca destas novas tecnologias.

1.2 Estado da Arte

A arquitetura de *software* utilizada ao longo da história passou por profundas revoluções. De um sistema multiusuário, passou–se a estruturas de duas e três camadas, evidenciando os sistemas de rede e, posteriormente os sistemas cliente/servidor evoluindo

para o que está sendo considerado hoje uma das novas tecnologias a serviço da interoperabilidade: os serviços web, que se constituem nos recursos necessários para se utilizar a internet como uma grande aplicação dinâmica.

Os serviços web, apresentam como pilares de seu sucesso 4 camadas conceituais (transporte, troca de mensagem, descrição de serviço e publicação de serviço) que lhe dão a possibilidade de realizar a interação dinâmica entre diferentes aplicações. Essas quatro camadas são representadas por tecnologias como UDDI, WSDL, SOAP, XML e HTTP (*Hypertext Transfer Protocol*).

O HTTP é o protocolo padrão da Internet e possui livre acesso na grande maioria dos firewalls, o que fez com que fosse adotado também como protocolo padrão de transporte dos serviços web.

O XML se constitui numa linguagem de marcação normalizada pelo consórcio da W3C (*World Wide Web Consortium*), destinada a transmitir dados via internet em forma de texto simples, o que possibilita a sua interoperabilidade entre todos os sistemas e plataformas.

O SOAP é o protocolo responsável pela troca de mensagens entre os serviços web. Constituem na realidade a implementação de RPC feitos de maneira padronizada e aberta, utilizando–se para isso da linguagem XML o que caracteriza o SOAP como um *framework* XML para transporte de mensagens entre aplicações.

O SOAP envia suas solicitações e recebe as respostas via o protocolo HTTP. Isso constituísse em uma grande vantagem uma vez que os servidores normalmente não mantêm *firewall* para aplicações HTTP. Por outro lado, esse também pode ser considerado um dos pontos falhos dos serviços web no que diz respeitos às questões de segurança. No entanto, já existem estudos e soluções iniciais com vistas a resolver as questões de segurança, que neste caso podem ser tratadas em dois níveis nos serviços web: em nível de transporte e em nível de aplicação. Iniciativas como o XKMS (XML *Key Management Specification*) e PKI (*Public Key Infraestruture*) estão sendo normatizadas por organizações como a W3C e prometem resolver problemas no que tange as questões de identificação e autenticação. A criptografia, os certificados digitais também estão sendo utilizadas para garantir a segurança necessária em nível de identificação, autenticação, autorização, integridade, privacidade, não–repúdio. A área de segurança em serviços web requer ainda muita pesquisa, contudo já é possível garantir uma certa segurança nos serviços prestados, como por exemplo, o uso de criptografia XML dentro dos documentos SOAP.

O WSDL apresenta a descrição da interface de comunicação do serviço web, ou seja, as instruções necessárias para a elaboração de um documento SOAP são descritas em um documento WSDL, onde são definidos como as chamadas devem ser invocadas, bem como demais detalhes a serem observados.

Para que as demais aplicações consigam acesso ao documento WSDL que expõe as regras de como acessar o serviço web com o uso de instruções SOAP, é necessário que essas descrições sejam publicadas em algum local público e de fácil acesso na *web*. Para isso existe a UDDI.

A UDDI representa o registro de todos os servicços web existentes devidamente publicados. Através da UDDI pode–se localizar os serviços disponíveis, acessar o documento WSDL do serviço e, invocá–lo conforme as normas ali descritas.

Na realidade esse conjunto de tecnologias XML, SOAP, WSDL, UDDI constituem–se num novo *middleware* para as aplicações hoje disponíveis na internet.

1.3 Objetivos do Trabalho

1.3.1 Objetivo geral

O presente trabalho tem como objetivo principal mostrar que a integração entre o paradigma de orientação a objetos e a arquitetura orientada a serviços (*serviços web*) é uma das principais formas de garantir a interoperabilidade e a reusabilidade de *softwares*, principalmente nas aplicações de internet.

1.3.2 Objetivos específicos

- Efetuar um levantamento das vantagens da orientação a objetos.
- Comprovar a evolução do paradigma de orientação a objetos.
- Mostrar que a estrutura adotada pelas principais soluções *middlewares* existentes até o advento dos *serviços web* estava voltada à orientação a objetos.
- Realizar uma pesquisa aprofundada sobre a arquitetura orientada a serviços e às tecnologias envolvidas (HTTP, XML, SOAP, UDDI, WSDL, eb-XML).
- Comparar a arquitetura orientada a serviços e arquitetura orientada a objetos, entendendo como se da a relação entre ambas.

- Efetuar levantamento das ferramentas e aplicações existentes para implementação dos *serviços web* e comprovar o uso do paradigma de orientação a objetos nas mesmas.

- Comprovar a necessidade do uso de padrões de tecnologia para a garantia da interoperabilidade.

- Mostrar que a interoperabilidade, a reusabilidade e a integração dos sistemas são possíveis e se constituem numa exigência para o desenvolvimento de aplicações, principalmente para as aplicações voltadas a internet.

- Compilar e divulgar as normas e estruturas funcionais contempladas pelas tecnologias dos *serviços web*.

1.4 Organização do Trabalho

O trabalho está estruturado de forma a dar uma rápida visão da evolução do paradigma de orientação a objetos e dos *middlewares*, detalhando posteriormente as tecnologias envolvidas com a arquitetura orientada a serviços, realizando as devidas comparações e conclusões a cerca do objetivo proposto.

O segundo capítulo apresenta de forma sucinta os conceitos sobre importantes áreas que se constituíram como base para a evolução da arquitetura orientada a serviços. O paradigma de orientação a objetos é abordado de forma direta, focando os aspectos principais, suas vantagens, os padrões de modelagem existentes e as principais linguagens de programação. A arquitetura dos principais *middlewares* é apresentada de forma sucinta, visando principalmente traçar um paralelo entre CORBA (*Common Object Request Broker Architetura)* DCOM *(Distributed Component Object Model)* e RMI (*Remote Method Invocation)*.

O terceiro capítulo visa detalhar os conceitos básicos dos serviços web, analisando a maneira como os mesmos são acessados, classificando–os em serviços web simples e serviços web complexos. Enfoca–se neste capítulo as camadas conceituais dos serviços web e seus relacionamentos, bem com a estrutura básica de troca de mensagens em XML.

Por se constituir no carro chefe da interoperabilidade dos serviços web, a XML é tratada de modo especial no capítulo quatro. Inicialmente tem–se uma visão histórica da XML, passando–se ao relato dos principais recursos e aplicabilidades desta ferramenta que está causando uma verdadeira revolução no mundo da Internet. O capítulo é concluído com

os avanços e pesquisas que estão sendo realizados nas diversas áreas, mas principalmente na área de segurança com o uso da XML.

O quinto capítulo apresenta o protocolo SOAP, mostrando sua origem, vantagens e desvantagem. A estrutura do *framework* SOAP é exposta de maneira clara, possibilitando um melhor entendimento do seu funcionamento.

O capítulo seis enfoca a descrição da interface dos serviços web que é realizada pela WSDL. São detalhados a especificação e os elementos individuais, bem como a forma de elaboração de um documento WSDL.

O registro dos serviços web é algo extremamente importante e, por conseguinte é abordado no capítulo sete. Os serviços web apresentam atualmente dois formatos de registro o UDDI e o eb–XML (*Eletronic Business eXtendet Markup Language)*. Ambos são detalhados e diferenciados no decorrer deste capítulo.

No oitavo capítulo faz–se uma abordagem especial sobre a questão da segurança nos serviços web. São expostos os principais problemas de segurança apresentados e as soluções nas quais estão sendo desenvolvidas pesquisas e normalizações. Enfoca–se neste capítulo principalmente a Criptografia XML e XKMS (*XML Key Management Specification*) que constituem–se em importantes iniciativas na busca de segurança nos serviços web.

Por fim, no nono capítulo são apresentadas às considerações finais do trabalho, onde são realizadas comparações entre a realidade possibilitada pelos serviços web e a contribuição das tecnologias apresentadas, conjuntamente com o paradigma de orientação a objetos. É enfocada de forma mais clara a relação entre a arquitetura orientada a objetos e a arquitetura orientada a serviços, comprovando–se que a vantagem oferecida por ambas gira em torno da interoperabilidade e da reusabilidade, mostrando–se a necessidade de que ambas sejam utilizadas de forma conjunta para que os serviços web atinjam os objetivos aos quais realmente se propõe, ou seja, a dinamicidade das aplicações.

2 REVOLUÇÃO DA INFORMAÇÃO: OO E MIDDLEWARES

A informática tem uma linha de vida extremamente curta, se comparada ao período que muitos outros inventos demoraram a se disseminar da mesma forma. No entanto essa linha de vida é extremamente dinâmica e fez com que o mundo das informações sofresse na realidade uma grande revolução. Dentro dessa idéia algumas tecnologias, ferramentas e paradigmas tiveram contribuições significativas.

2.1 Paradigma de Orientação a Objetos

Ferramentas de desenvolvimento de *softwares* estão presentes no cotidiano da informática há mais de seis décadas. Desde o seu surgimento até os dias de hoje, o modo de pensar e desenvolver sistemas aplicativos evoluiu de forma gigantesca.

Seu primórdio originou–se no desenvolvimento denominado estruturado, onde os sistemas possuíam um baixo grau de análise e um alto grau de programação, tornando–os suscetíveis a falhas, as quais demandam várias horas de re–trabalho do programador.

Na década de sessenta uma nova filosofia começou a ser elaborada e implementada por alguns programadores da época. Esta se denominou paradigma orientado a objeto, a qual agrega alto grau de análise e menor nível de programação.

A OO (Orientação a Objetos) é projetada para permitir a definição dos objetos que compõe os programas e das propriedades que estes objetos contém. Objetos e mensagens são os mecanismos centrais da Tecnologia OO. Logo: “OO é olhar como as coisas são antes de saber como elas funcionam”. MARTIM (07)

A OO surgiu como um novo paradigma de desenvolvimento que trazia inúmeras vantagens, como a reutilização e a facilidade de manutenção, por exemplo.

A OO não dividia mais o problema em dados e procedimentos, como no modelo estruturado, mas sim em objetos englobando ambos. Então, por falta de um método e ferramentas adequadas para estimular um pensamento orientado a objetos, os benefícios da OO não eram atingidos conforme o esperado. É um erro muito comum achar que utilizar uma linguagem de programação OO é programar orientado a objetos. MARTIM (07)

Os conceitos OO começaram a se solidificar nos anos 60/70 – sistemas comerciais tornaram–se disponíveis na segunda metade dos anos 80, quando *SmallTalk* e C++ tornaram–se realidade, vindo logo a seguir os bancos de dados e as técnicas OO, das quais há hoje uma grande variedade. Pode–se dizer que o termo OO foi criado nos anos 70, durante o desenvolvimento do *SmallTalk*, no centro de pesquisas da Xerox em Palo Alto, Califórnia. MARTIM (07)

2.1.1 Programação orientada a objeto

As linguagens de programação vêm apresentando uma grande evolução ao longo dos anos, inicialmente o que se pode comprovar claramente é uma evolução caracterizada por décadas:

Anos 70 – Época da não estruturada, dados e códigos emaranhados.

Anos 80 – Época da estruturada, dados separados de códigos (modulares).

Anos 90 – Época da OO, dados e códigos organizados em objetos.

A partir dos anos 80, as principais linguagens passaram a incluir alguns conceitos de OO, como por exemplo, Pascal, C, *Lisp*, *Cobol*, depois evoluíram com a inclusão de classes. C++ foi um marco para aplicações em todos os níveis. Surge o Visual *Object Oriented Cobol (c/ win95*) e o Visual *Basic*. As linguagens mais utilizadas no mundo até a década de 90, segundo informação do *Richard Soley* e do *Jon Siegel* da OMG, em dezembro 96, são Cobol e Visual *Basic.*(12)

Hoje, já são mais de 2000 as linguagens de programação de alto nível diferentes. Dessas, cerca de cem são orientadas a objeto ou baseadas em objetos (uma linguagem é considerada orientada a objetos quando não só incorpora os conceitos de objetos como também suporta polimorfismo e herança). A grande distinção entre essas linguagens, no entanto, refere–se à capacidade de suportar as diversas características distintas que a OO possui, as quais a torna uma poderosa ferramenta de modelagem e desenvolvimento de *software*, como por exemplo, metaclasses, polimorfismo, variáveis e métodos de classe, encapsulamento e herança.

A figura 2.1 expressa a evolução das linguagens de programação, dando uma idéia da caminhada das linguagens OO.

A seguir descreve–se sucintamente a evolução das principais linguagens e metodologias que contribuíram para a Orientação a Objeto:

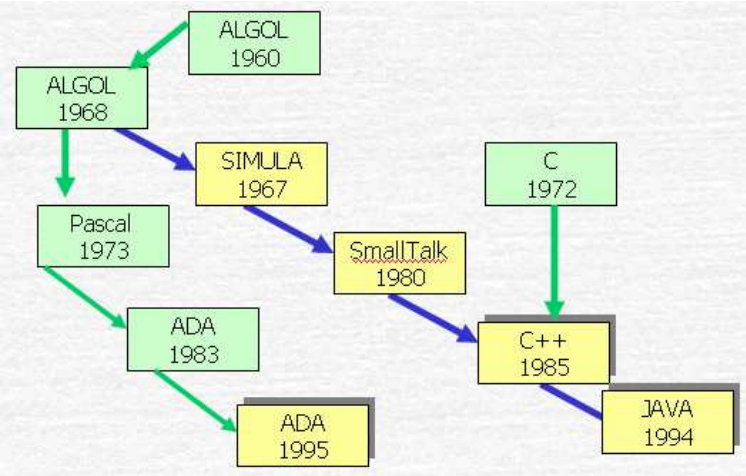

Figura 2.1 Evolução das Linguagens de Programação

2.1.1.1 ALGOL 60 *(Algorithmic Oriented Language*)

Linguagem algébrica de origem européia, desenvolvida pelo Comitê Internacional popular, destinada à resolução de problemas científicos. Devido às suas características inovadoras (nessa época) influenciou o projeto de quase todas as linguagens projetadas a partir de 1960. Descrita em BNF (*Backus–Naur Form*), foi projetada independentemente da implementação, o que permite uma maior criatividade e implementação mais difícil. É pouco usada em aplicações comerciais devido à ausência de facilidades de E/S e pelo pouco interesse de vendedores.

2.1.1.2 ALGOL 68

É muito diferente do ALGOL 60. Linguagem de propósito geral que foi projetada para a comunicação de algoritmos, para sua execução eficiente em vários computadores e para ajudar seu ensino a estudantes. A versão ALGOL 68 constitui a versão mais completa e mais moderna, na qual o conjunto das instruções possíveis constitui uma linguagem de âmbito geral. MARTIM (07)

2.1.1.3 SIMULA 67

Linguagem baseada em ALGOL 60, criada no início dos anos 60 por *Ole Johan Dahl e Kristan Nygaard*, na Noruega.

É a primeira linguagem a incorporar facilidades para definir classes de objetos genéricos na forma de uma hierarquia de classes e sub–classes. O Simula foi idealizada em 1966, na Noruega, como uma extensão da linguagem ALGOL 60. Uma classe em Simula é um módulo englobando a definição da estrutura e o comportamento comum a todas as suas instâncias (objetos). Como o nome indica, é uma linguagem adequada à programação de simulações de sistemas que podem ser modelados pela interação de um grande número de objetos distintos. As idéias de Simula serviram de base para as propostas de utilização de tipos abstratos de dados, como para o *SmallTalk*.

2.1.1.4 PASCAL

Desenvolvida por *Niklaus Wirth* em 1969, é uma linguagem de fácil aprendizagem e implementação, suportando programação estruturada. A natureza da linguagem força os programadores a desenvolverem os programas metodicamente e cuidadosamente, por esta razão é que esta linguagem é bastante popular para o ensino da programação. Contrariamente ao sucesso no ensino, o Pascal tem tido um modesto sucesso no mundo dos negócios. Parte da resistência ao Pascal por parte dos programadores profissionais tem origem na sua inflexibilidade e carência de ferramentas para desenvolvimento de grandes aplicações.

A linguagem PASCAL baseia–se num pequeno número de utensílios de programação e de estrutura de dados. A escolha destes utensílios efetua–se considerando propriedades semânticas que podem ser, facilmente, controladas durante a escrita e a compilação de programas. Desta forma, PASCAL oferece, além das estruturas de dados habituais, a noção de conjunto de objetos, permitindo considerar agrupamentos de objetos não ordenados. MARTIM (07)

2.1.1.5 SMALLTALK

A linguagem *SmallTalk* foi desenvolvida no Centro de Pesquisas da Xerox durante a década de 70.

SmallTalk é uma linguagem baseada em classes de objetos. Um objeto é uma variável que pode ser acessada somente através de operações associadas a ele. *SmallTalk*

é um exemplo de uma linguagem que segue o paradigma de orientação a objeto possuindo o conceito de classe do SIMULA 67 de mais encapsulamento, herança e instanciação.

A *SmallTalk* tem sido usada em diversos tipos de aplicações onde o ênfase está na Simulação de modelos de sistemas, como automação de escritórios, animação gráfica, informática educativa, instrumentos virtuais, editores de texto e bases de dados genéricos. Tais aplicações diferem substancialmente daquelas em que a ênfase está na resolução de problemas através de algoritmos, tais como problemas de busca, otimização e resolução numérica de equações.

A *SmallTalk* foi a primeira linguagem de programação a utilizar o paradigma de programação interativa, introduz o conceito de linguagem de programação extensível. MARTIM (07)

A linguagem evoluiu por décadas. Existem cinco releases claramente identificáveis: Smalltalk 72, 74, 76, 78, e a mais recente 80. As duas primeiras ainda não continham herança, mas já se falava em mensagens e polimorfismo. As demais foram completando as anteriores, até chegar ao ponto em que qualquer elemento do ambiente pode ser tratado como um objeto. Há ainda um importante dialeto do *Smalltalk*, da *Digitalk*: o *Smalltalk/V*, muito similar ao 80 e disponível em ambiente IBM–PC e Macintosh. Sua biblioteca de classes é muito semelhante, exceto pelas classes de interface com o usuário.

2.1.1.6 C

É uma linguagem de programação de alto–nível, desenvolvida pelo Bell Lab no início dos anos 70, visando a implementação do UNIX. Tem facilidades para a programação em "baixo nível" e gera código eficiente. Possui um grande conjunto de operadores, o que permite um código compacto, porém de baixa legibilidade. Embora originalmente desenvolvido como uma linguagem de programação de sistemas, o C provou ser uma linguagem poderosa e flexível que pode ser usada para uma variedade de aplicações. C é uma linguagem para programação de computadores pessoais – requerendo menos memória do que outras linguagens. O primeiro grande programa escrito em C foi o sistema operacional Unix, como também outras linguagens de programação, entre elas a linguagem C++ e Java, e por muitos anos C foi considerado bastante ligado com o Unix. Atualmente, C é uma linguagem independente do Unix.

Embora ela seja uma linguagem de alto nível, o C está mais perto das linguagens *Assembly* da maioria das outras linguagens de alto nível, pois possui muitas

instruções de baixo nível. Esta proximidade às linguagens máquina, permite aos programadores de C escreverem código bastante eficiente (rápido). MARTIM (07)

2.1.1.7 ADA

É uma linguagem de alto nível desenvolvida no fim dos anos 70 e princípios dos anos 80 pelo Departamento de Defesa dos Estados Unidos. É dedicada aos *"embedded systems*" (operam como parte de um sistema maior) e que se baseia no Pascal.

Esta linguagem surgiu quando o Departamento de Defesa dos Estados Unidos decidiu padronizar uma única linguagem de programação, visto os programadores que escreviam as aplicações militares utilizarem cerca de 500 linguagens de programação diferentes, o que tornava difícil de coordenar projetos e compatibilidade entre aplicações.

Teve um padrão em 1983. Além disso, usa conceitos de classe do Simula 67, adota o tratamento de exceções de PL/I e possui facilidades para processamento concorrente. Um das principais características desta linguagem é o suporte a aplicações de tempo–real. Em acréscimo, a ADA incorpora técnicas modulares que cria, para construir e manutenção de grandes sistemas. A ADA foi amplamente utilizada no desenvolvimento de linguagens para a maioria das aplicações militares dos U.S.A. MARTIM (07)

2.1.1.8 C++

É uma linguagem de programação de alto nível desenvolvida por *Bjarne Stroustrup* nos Laboratórios Bell, que pode ser vista também como uma extensão do C (o que pode ser ou não discutível). Seu ancestral imediato era o C *with* Classes, desenvolvido também por *Stroustrup* em 1980, e que por sua vez era baseado no C e no Simula.

C++ é uma versão OO do C, com o qual é compatível. Na realidade, pode ser visto como um superset do C. Código em C pode ser incorporado a programas C++. É uma linguagem que produz programas rápidos, eficientes, ao custo de sacrificar alguma flexibilidade. C ++ tornou–se popular a ponto de fazer com que muitos novos compiladores C sejam na realidade C/C++.

O C++ permite herança simples e múltipla. Na primeira, uma classe dá origem a uma ou várias, mas cada uma tem apenas um antecedente imediato.

2.1.1.9 JAVA

A linguagem Java é uma linguagem de terceira geração. É uma linguagem OO de uso geral com características próprias para se espalhar via Internet.

Em 1991 um grupo de cientistas da SUN a criou (projeto Green). O objetivo era atender empresas de eletrônica de consumo. Em setembro de 1994 percebeu–se que tinha tudo a ver com a *web*. Arthur Van Hoff reescreveu então o compilador Java originalmente escrito em C por *James Gosling*, na própria linguagem Java.

A arquitetura do Java não é nem radical nem especialmente nova. Resumindo, os aplicativos em Java são compilados em um código de *bytes* independente de arquitetura. Esse código de *bytes* pode então ser executado em qualquer plataforma que suporte um interpretador Java. O Java requer somente uma fonte e um binário e, mesmo assim, é capaz de funcionar em diversas plataformas.

Na linguagem Java há uma biblioteca de procedimentos TCP/IP incluída nos códigos–fonte e de distribuição binária do Java. Isso facilita aos programadores os acessos remotos às informações, usando protocolos como HTTP e FTP.

Os programas do Java são compilados em formato binário de código de *bytes*, que então são interpretados por um ambiente de execução do Java específico da plataforma em questão. Portanto é ao mesmo tempo compilado e interpretado.

O compilador Java foi escrito com o próprio Java, enquanto seu ambiente de tempo e execução foi escrito em ANSI C e tem uma interface de portabilidade bem definida e concisa.

Os objetos binários de códigos de bytes do Java são formados por seqüências de execução múltiplas e simultâneas. Essas seqüências são conhecidas como contextos de execução ou processos leves. As linguagens C C++ são membros de um paradigma de execução em seqüência única, por não oferecerem suporte a seqüências no nível de linguagem. O Java, no entanto, oferece suporte no nível de linguagem para multitarefa, resultando em uma abordagem de programação mais poderosa e de múltiplas facetas.

O projeto dinâmico permite que os programas Java se adaptem aos ambientes computacionais mutantes. Por exemplo, a maior parte dos desenvolvimentos típicos em C++ se baseia muito em bibliotecas de classe que podem ser de propriedade desenvolvida por terceiros. Muitas bibliotecas de terceiros, como as distribuídas com sistemas operacionais ou sistemas de janelas, são *linkeditadas* dinamicamente e vendidas ou distribuídas

separadamente dos aplicativos que delas dependam. Quando essas bibliotecas são atualizadas, os aplicativos que dependerem dela poderão apresentar problemas, até que sejam recompilados e redistribuídos. Isso adiciona mais um custo à manutenção do software.

Como o Java foi criado para ambientes de rede, os recursos de segurança receberam muita atenção. Por exemplo, ao se executar um binário transferido por *download* da rede, o mesmo poderá estar infectado por vírus. Os aplicativos Java apresentam garantia de resistência contra vírus e de que não são infectados por vírus, pois não são capazes de acessar *heaps*, *stacks* ou memória do sistema. No Java, a autenticação do usuário é implementada com um método de chave pública de criptografia. Isso impede de maneira eficaz que *hackers* e *crakers* examinem informações protegidas como nomes e senhas de contas.

Um dos principais objetivos do projeto do Java foi criar uma linguagem o mais próxima possível do C ++, para garantir sua rápida aceitação no mundo do desenvolvimento OO. Outro objetivo do seu projeto foi eliminar os recursos obscuros e danosos do C ++, que fugiam à compreensão e aumentavam a confusão que poderia ocorrer durante as fases de desenvolvimento, implementação e manutenção do *software*. O Java é simples porque é pequeno. O interpretador básico do Java ocupa aproximadamente 40k de RAM, excluindo–se o suporte a multitarefa as bibliotecas padrão, que ocupam outros 175k. Mesmo a memória combinada de todos esses elementos é insignificante, se comparada a outras linguagens e ambientes de programação.

Há muitas situações em que a interpretação de objetos de códigos de *bytes* proporciona desempenho aceitável. Mas outras circunstâncias exigem desempenhos mais altos. O Java concilia tudo isso oferecendo a tradução dos códigos de *bytes* para o código de máquina nativo em tempo de execução.

O Java é uma linguagem de programação desenvolvida pela *Sun Microsystems* e tinha como proposta inicial ser aplicado em pequenos aparelhos eletrônicos. Hoje Java é a linguagem que mais se fortalece com a Internet, pois é uma linguagem simples, orientada a objetos, distribuída, interpretada, robusta, segura, independente de arquitetura, portável, de alto desempenho multitarefa e dinâmica, que oferece inúmeros pacotes e API's dentro das mais novas tecnologias, aqui enquadrando–se os recursos dos *serviços web* que serão detalhados nos próximos capítulos.

2.1.1.10 Object pascal

O *Object* Pascal foi criado pelos desenvolvedores da *Apple Computer* (alguns dos quais envolvidos no desenvolvimento do *SmallTalk*), em conjunto com *Niklaus Wirth*, o designer do Pascal. Seu ancestral imediato é o Clascal, uma versão orientada a objetos do Pascal para o Lisa.

O *Object* Pascal foi tornado público em 1986, e foi a primeira linguagem orientada a objetos suportada pela *Macintosh Programmer's Workshop* (MPW), o ambiente de desenvolvimento da família *Macintosh*. A biblioteca de classes para a MPW, chamada MacApp, provê um ambiente para construção de aplicações que se adequam às normas de interface gráfica do Mac.

2.1.1.11 Delphi

Delphi é um ambiente de desenvolvimento de aplicações para *Windows* criado pela *Borland* e baseado no conceito de componentes, classes pré–fabricadas e, em boa parte das vezes, visuais. Suas ferramentas de design possibilitam rápida criação de protótipos de aplicações, que somados ao amplo conjunto de seus componentes, tornam os protótipos em aplicações robustas e bastante otimizadas.

A linguagem *Delphi* tem como base o *Object* Pascal e conta ainda com alguns outros recursos que foram incorporados, como por exemplo: método de classe, ancestral *default*, declaração de classes com forward, ponteiros para métodos, referências a classe (metaclasses).

2.1.1.12 Eiffel

O Eiffel foi criado pela ISE (*Eiffel Software Inc*.), é tratado como uma linguagem totalmente orientada a objetos, procurando adequar todos os conceitos OO em uma linguagem de código eficiente. A linguagem permite encapsulamento, herança, polimorfismo. (05)

2.1.2 Modelo conceitual da orientação a objetos

O modelo de objetos permite a criação de bibliotecas que tornam efetivos o compartilhamento e a reutilização de código, reduzindo o tempo de desenvolvimento e, principalmente, simplificando o processo de manutenção das aplicações.

A grande dificuldade para compreender a programação OO é a diferença de abordagem do problema. Enquanto a programação estruturada tem como principal foco as

ações (procedimentos e funções), a programação OO se preocupa com os objetos e seus relacionamentos. Além do conceito de objeto, a programação OO tem como alicerces os conceitos de encapsulamento, classe, herança e polimorfismo.

Programação OO	Programação Estruturada
Métodos	Procedimentos e funções
Instâncias de variáveis	Variáveis
Mensagens	Chamadas a procedimentos e funções
Classes	Tipos de dados definidos pelo usuário
Herança	–
Polimorfismo	–

Quadro 2.1 – Comparativo entre programação estruturada e OO

2.1.2.1 Objeto

Um objeto é uma abstração de *software* que pode representar algo real ou virtual. Um objeto é formado por um conjunto de propriedades (variáveis) e procedimentos (métodos). As variáveis possuem um tipo, que define os possíveis valores que a variável pode representar, como um número inteiro, número real ou *string*. Os métodos são rotinas que, quando executadas, realizam alguma tarefa, como alterar o conteúdo de uma variável do objeto.

Os objetos se comunicam apenas através de *mensagens*, como se pode verificar na figura 2.2 Quando um objeto deseja alguma tarefa de um outro objeto, ele envia uma mensagem contendo o nome do objeto–origem, nome do objeto–destino, nome do método a ser ativado no objeto–destino e, se necessário, parâmetros que permitem especificar alguma função especial a ser executada pelo método. Este conceito se assemelha à chamada de uma rotina em uma linguagem tradicional. O conjunto de mensagens que um objeto pode responder é definido como protocolo de comunicação.

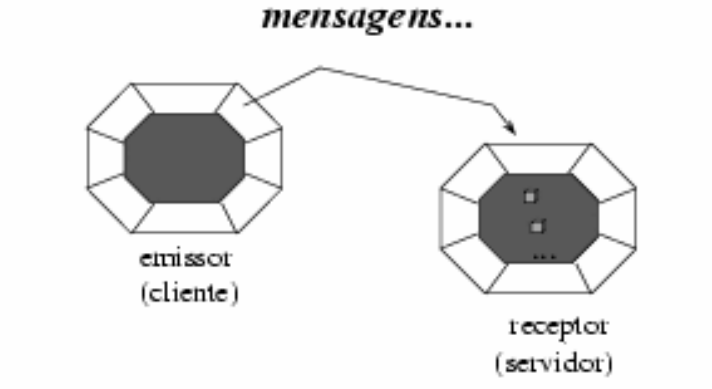

Figura 2.2 Comunicação entre objetos

As variáveis de um objeto só podem ser alteradas por métodos definidos na própria classe. A única maneira de um objeto alterar as variáveis de um outro objeto é a através da ativação de um de seus métodos por uma mensagem. Este conceito, onde variáveis e métodos são visíveis apenas através de mensagens, é conhecido como *encapsulamento*. O encapsulamento funciona como uma proteção para as variáveis e métodos, além de tornar explícito qualquer tipo de comunicação com o objeto.

Todos os objetos possuem um estado que significa o resultado de atividades executadas pelo objeto, e é normalmente determinada pelos valores de seus atributos e ligações com outros objetos.

Um objeto muda de estado quando acontece algo, o fato de acontecer alguma coisa com o objeto é chamado de evento. Através da análise da mudança de estados dos tipos de objetos de um sistema, podem-se prever todos os possíveis comportamentos de um objeto de acordo com os eventos que o mesmo possa sofrer. O tratamento de evento na orientação a objetos dá–se pela caracterização de novo grupo de objetos, ou seja, uma nova classe.

2.1.2.2 Classe

Uma classe consiste de variáveis e métodos que representam características de um conjunto de objetos semelhantes. O conceito de classe é um dos pilares da programação orientada a objetos, por permitir a reutilização efetiva de código.

Uma classe pode ser a descrição de um objeto em qualquer tipo de sistema – sistemas de informação, técnicos, integrados real–time, distribuídos, *software*. Num sistema de *software*, por exemplo, existem classes que representam entidades de *software* num sistema operacional como arquivos, programas executáveis, janelas, barras de rolagem, entre outros.

A declaração de uma classe é similar ao dos tipos definidos pelo usuário nas linguagens de programação de alto nível. Nestas linguagens é possível definir um novo tipo de dado e declarar uma variável deste tipo. No caso de objetos, primeiro define–se uma classe com suas variáveis e métodos e, depois se declara um objeto desta nova classe. Um objeto é definido como sendo uma instância de uma determinada classe. A classe é estática, enquanto o objeto é dinâmico.

2.1.2.3 Pacotes

Pacote é um mecanismo de agrupamento, onde todos os modelos de elementos podem ser agrupados. Todos os modelos de elementos que são ligados ou referenciados por um pacote são chamados de conteúdo do pacote. Um pacote possui vários modelos de elementos, e isto significa que estes não podem ser incluídos em outros pacotes.

O pacote tem uma grande similaridade com a agregação (relacionamento que será tratado em seguida). O fato de um pacote ser composto de modelos de elementos cria uma agregação de composição. Se este for destruído, todo o seu conteúdo também será.

2.1.2.4 Componentes

Um componente pode ser tanto um código em linguagem de programação como um código executável já compilado. Por exemplo, em um sistema desenvolvido em Java, cada arquivo *.java* ou *.class* é um componente do sistema

2.1.3 Análise orientada a objetos

A análise orientada a objetos inicialmente foi constituída de conceitos que eram considerados inovadores comparados com os outros métodos. Isto implicava em uma mudança radical sobre as metodologias orientadas ao processo (como a análise estruturada), porém, produzindo somente uma mudança incremental sobre as metodologias orientadas aos dados. De fato, seus conceitos podiam parecer diferentes, porém eles são muito naturais e se aplicam ao nosso mundo real.

O objetivo da análise orientada a objetos é o de desenvolver uma série de modelos de análise, satisfazendo um conjunto de requisitos definidos pelo problema em si. Isto é, esses modelos descrevem, de forma estruturada, as informações, as funções e o comportamento de seus elementos constituintes. A informação (ou modelo de objeto) contém a definição dos objetos no sistema, os quais incluem: o nome do objeto, os atributos do objeto, e os relacionamentos que um objeto tem com outros. O modelo de comportamento (ou de estado) descreve o comportamento dos objetos em termos das transições permitidas entre os objetos e os eventos que causam a mudança de estado desses objetos.

Uma grande variedade de métodos de análise orientada a objetos foram desenvolvidos desde 1988. Porém, todos eles possuem características comuns entre si, quais sejam:

- Representação de classes e hierarquias;
- Criação de modelos de relacionamento de objetos;
- Derivação de modelos de comportamento de objetos;

A análise orientada a objetos possui muitas vantagens:

- Manutenibilidade através da simplificação do mapeamento do mundo real, o qual proporciona menos esforço de análise, menos complexidade no design do sistema, e uma fácil verificação por parte do usuário.
- Reusabilidade dos artifícios da análise, os quais economizam tempo e custos.
- Ganhos na produtividade através do mapeamento direto pelas características das Linguagens de Programação Orientadas a Objetos.

2.1.4 Modelagem orientada a objeto

Os conceitos da orientação a objetos já vêm sidos discutidos há muito tempo, desde o lançamento da 1ª linguagem orientada a objetos, a SIMULA. Vários pesquisadores da engenharia de *software* mundial como *Peter Coad, Edward Yourdon* e Roger Pressman abordaram extensamente a análise orientada a objetos como realmente um grande avanço no desenvolvimento de sistemas. Mas mesmo assim, eles citam que não existe (ou que não existia no momento de suas publicações) uma linguagem que possibilitasse o desenvolvimento de qualquer software utilizando a análise orientada a objetos.

Os conceitos que *Coad, Yourdon, Pressman* e tantos outros abordaram, discutiram e definiram em suas publicações foram que:

- A orientação a objetos é uma tecnologia para a produção de modelos que especifiquem o domínio do problema de um sistema.
- Quando construídos corretamente, sistemas orientados a objetos são flexíveis a mudanças, possuem estruturas bem conhecidas e provêm a oportunidade de criar e implementar componentes totalmente reutilizáveis.
- Modelos orientados a objetos são implementados convenientemente utilizando uma linguagem de programação orientada a objetos. A engenharia de software orientada a objetos é muito mais que utilizar mecanismos de sua linguagem de

programação, é saber utilizar a melhor forma possível todas as técnicas da modelagem orientada a objetos.

- A orientação a objetos não é só teoria, mas uma tecnologia de eficiência e qualidade comprovada usada em inúmeros projetos e para construção de diferentes tipos de sistemas.

Nesta caminhada, *Jacobson, Booch, Rambaugch, Coad, Yourdan* e outros nomes importantes ligados ao paradigma da orientação a objetos apresentaram várias metodologias para o desenvolvimento das fases de análise e projeto de sistemas em OO. Para alguns estas fases deveriam ser tratadas de forma totalmente distintas e, para outros ambas constituíam–se num conjunto separado apenas pelo nível de abstração.

A UML (*Unified Modeling Language)* é uma tentativa de padronizar a modelagem orientada a objetos de uma forma que qualquer sistema, seja qual for o tipo, possa ser modelado corretamente, com consistência, fácil de se comunicar com outras aplicações, simples de ser atualizado e compreensível.

Existem várias metodologias de modelagem orientada a objetos que até o surgimento da UML causavam uma guerra entre a comunidade de desenvolvedores orientado a objetos. A UML acabou com esta guerra trazendo as melhores idéias de cada uma destas metodologias, e mostrando como deveria ser a migração de cada uma para a UML, definindo também as regras a serem utilizadas e clarificando as fases de análise e projeto.

2.1.4.1 UML

A UML é o sucessor de um conjunto de métodos de análise e projeto orientados a objeto (OOA&D). Foi desenvolvidas por *Grady Booch, James Rumbaugh, e Ivar* Jacobson que possuem um extenso conhecimento na área de modelagem orientado a objetos já que as três mais conceituadas metodologias de modelagem orientada a objetos foram eles que desenvolveram e a UML é a junção do que havia de melhor nestas três metodologias adicionados novos conceitos e visões da linguagem.

Em 1997, a UML versão 1.1 foi submetida a OMG (*Object Management Group*) para padronização.

A OMG é uma organização internacional, fundada em 1989, formada por 800 membros incluindo diversas empresas ligadas ao uso e dissemenicação da tecnologia da

informação. A OMG promove a teoria e a prática da tecnologia orientada a objetos em desenvolvimento de sistemas.

A UML é uma linguagem para especificação, visualização, construção e documentação de modelos de sistemas de *software* e também pode ser utilizada para modelagem de negócios. Ela é usada desde a especificação da análise de requisitos até a finalização com a fase de testes.

A especificação da UML submetida para aprovação da OMG não tem as características de um método, pois ela não apresenta uma seqüência de passos para a resolução de um problema, como deveria ser no caso de um método. FOWLER (06) UML(02)

Segundo a UML versão 1.3 (junho de 99), são objetivos:

a) Prover os usuários com uma linguagem de modelagem visual, pronta para uso, para o desenvolvimento de modelos significativos;

b) Pornecer mecanismos de ampliação e especialização que possam estender os conceitos principais;

c) Ser independente de linguagens de programação e processos de desenvolvimento;

d) Prover uma base formal para entendimento da linguagem de modelagem;

e) Incentivar o crescimento do mercado de ferramentas orientadas a objeto;

f) Suportar conceitos de desenvolvimento de alto nível, tais como colaboração, *frameworks*, padrões e componentes;

g) Reunir as melhores práticas de modelagem.

A UML define uma notação e uma meta–modelo. A notação são todos os elementos de representação gráfica vistos no modelo (retângulo, setas, o texto), é a sintaxe do modelo de linguagem. A notação do diagrama de classes, por exemplo, define a representação de itens e conceitos tais como: classe, associação e multiplicidade. Uma meta–modelo é um diagrama de classe que define de maneira mais rigorosa a notação.

Através da UML pode–se descrever qualquer tipo de sistema, em termos de diagramas orientado a objetos. Naturalmente, o uso mais comum é para criar modelos de sistemas de *software*, mas a UML também é usada para representar sistemas mecânicos sem nenhum *software*. Aqui estão alguns tipos diferentes de sistemas com suas características mais comuns:

- Sistemas de Informação: Armazenar, pesquisar, editar e mostrar informações para os usuários. Manter grandes quantidades de dados com relacionamentos complexos, que são guardados em bancos de dados relacionais ou orientados a objetos.

- Sistemas Técnicos: Manter e controlar equipamentos técnicos como de telecomunicações, equipamentos militares ou processos industriais. Eles devem possuir interfaces especiais do equipamento e menos programação de *software* de que os sistemas de informação. Sistemas Técnicos são geralmente sistemas real–time.

- Sistemas *Real–time* Integrados: Executados em simples peças de *hardware*s integrados a telefones celulares, carros, alarmes e outros. Estes sistemas implementam programação de baixo nível e requerem suporte *real–time*.

- Sistemas Distribuídos: Distribuídos em máquinas onde os dados são transferidos facilmente de uma máquina para outra. Eles requerem mecanismos de comunicação sincronizados para garantir a integridade dos dados e geralmente são construídos em mecanismos de objetos como CORBA, COM/DCOM ou Java Beans/RMI.

- Sistemas de *Software*: Definem uma infra–estrutura técnica que outros *softwares* utilizam. Sistemas Operacionais, bancos de dados, e ações de usuários que executam ações de baixo nível no *hardware*, ao mesmo tempo em que disponibilizam interfaces genéricas de uso de outros *softwares*.

- Sistemas de Negócios: descreve os objetivos, especificações (pessoas, computadores), as regras (leis, estratégias de negócios), e o atual trabalho desempenhado nos processos do negócio.

A maioria dos sistemas não possui apenas uma destas características acima relacionadas, mas várias delas ao mesmo tempo. Sistemas de informações de hoje, por

exemplo, podem ter tanto características distribuídas como *real–time*. E a UML suporta modelagens de todos estes tipos de sistemas.

A UML aborda o caráter estático e dinâmico do sistema a ser analisado levando em consideração, já no período de modelagem, todas as futuras características do sistema em relação à utilização de *packages* próprios da linguagem a ser utilizada, utilização do banco de dados bem como as diversas especificações do sistema a ser desenvolvido de acordo com as métricas finais do sistema. Para tanto se utilizam diversos diagramas.

Os diagramas utilizados pela UML são compostos de nove tipos: diagrama de use case, de classes, de objeto, de estado, de seqüência, de colaboração, de atividade, de componente e o de execução.

Todos os sistemas possuem uma estrutura estática e um comportamento dinâmico. A UML suporta modelos estáticos (estrutura estática), dinâmicos (comportamento dinâmico) e funcional. A Modelagem estática é suportada pelo diagrama de classes e de objetos, que consiste nas classes e seus relacionamentos. Os relacionamentos podem ser de associações, herança (generalização), dependência ou refinamentos. Os modelos dinâmicos são suportados pelos diagramas de estado, seqüência, colaboração e atividade. E o modelo funcional é suportado pelos diagramas de componente e execução. FOWLER (06) UML(02)

a) Diagrama use–case

A modelagem de um diagrama use–case é uma técnica usada para descrever e definir os requisitos funcionais de um sistema. Eles são escritos em termos de atores externos, use–cases e o sistema modelado. Os atores representam o papel de uma entidade externa ao sistema como um usuário, um *hardware*, ou outro sistema que interage com o sistema modelado. Os atores iniciam a comunicação com o sistema através dos use–cases, onde o use–case representa uma seqüência de ações executadas pelo sistema e recebe do ator que lhe utiliza dados tangíveis de um tipo ou formato já conhecido, e o valor de resposta da execução de um use–case (conteúdo) também já é de um tipo conhecido, tudo isso é definido juntamente com o use–case através de texto de documentação.

b) Diagrama de classes

O diagrama de classes demonstra a estrutura estática das classes de um sistema onde estas representam os objetos que são gerenciados pela aplicação modelada. Classes podem se relacionar com outras através de diversas maneiras: associação

(conectadas entre si), dependência (uma classe depende ou usa outra classe), especialização (uma classe é uma especialização de outra classe), ou em pacotes (classes agrupadas por características similares). Todos estes relacionamentos são mostrados no diagrama de classes juntamente com as suas estruturas internas, que são os atributos e operações.

O diagrama de classes é considerado estático já que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema. Um sistema normalmente possui alguns diagramas de classes, já que não são todas as classes que estão inseridas em um único diagrama e uma certa classe pode participar de vários diagramas de classes.

Uma classe num diagrama pode ser diretamente implementada utilizando–se uma linguagem de programação orientada a objetos que tenha suporte direto para construção de classes. Para criar um diagrama de classes, as classes têm que estar identificadas, descritas e relacionadas entre si.

c) Diagrama de objetos

O diagrama de objetos é uma variação do diagrama de classes e utiliza quase a mesma notação. A diferença é que o diagrama de objetos mostra os objetos que foram instanciados das classes.

Os diagramas de objetos não são tão importantes como os diagramas de classes, mas eles são muito úteis para exemplificar diagramas complexos de classes ajudando muito em sua compreensão. Diagramas de objetos também são usados como parte dos diagramas de colaboração, onde a colaboração dinâmica entre os objetos do sistema é mostrada.

d) Diagrama de estado

O diagrama de estado é tipicamente um complemento para a descrição das classes. Este diagrama mostra todos os estados possíveis que objetos de uma certa classe podem se encontrar e mostra também quais são os eventos do sistema que provocam tais mudanças.

Os diagramas de estado não são escritos para todas as classes de um sistema, mas apenas para aquelas que possuem um número definido de estados conhecidos e onde o comportamento das classes é afetado e modificado pelos diferentes estados.

Diagramas de estado capturam o ciclo de vida dos objetos, subsistemas e sistemas. Eles mostram os estados que um objeto pode possuir e como os eventos (mensagens recebidas, timer, erros, e condições sendo satisfeitas) afetam estes estados ao passar do tempo.

e) Diagrama de seqüência

Um diagrama de seqüência mostra a colaboração dinâmica entre os vários objetos de um sistema. O mais importante aspecto deste diagrama é que a partir dele percebe–se a seqüência de mensagens enviadas entre os objetos. Ele mostra a interação entre os objetos, alguma coisa que acontecerá em um ponto específico da execução do sistema.

O diagrama de seqüência consiste em um número de objetos mostrado em linhas verticais. O decorrer do tempo é visualizado observando–se o diagrama no sentido vertical de cima para baixo. As mensagens enviadas por cada objeto são simbolizadas por setas entre os objetos que se relacionam.

Diagramas de seqüência possuem dois eixos: o eixo vertical, que mostra o tempo e o eixo horizontal, que mostra os objetos envolvidos na seqüência de uma certa atividade. Eles também mostram as interações para um cenário específico de uma certa atividade do sistema.

f) Diagrama de colaboração

Um diagrama de colaboração mostra de maneira semelhante ao diagrama de seqüência, a colaboração dinâmica entre os objetos. Normalmente pode–se escolher entre utilizar o diagrama de colaboração ou o diagrama de seqüência.

No diagrama de colaboração, além de mostrar a troca de mensagens entre os objetos, percebem–se também os objetos com os seus relacionamentos. A interação de mensagens é mostrada em ambos os diagramas. Se a ênfase do diagrama for o decorrer do tempo, é melhor escolher o diagrama de seqüência, mas se a ênfase for o contexto do sistema, é melhor dar prioridade ao diagrama de colaboração.

g) Diagrama de atividade

Diagramas de atividade capturam ações e seus resultados. Eles focam o trabalho executado na implementação de uma operação (método), e suas atividades numa

instância de um objeto. O diagrama de atividade é uma variação do diagrama de estado e possui um propósito um pouco diferente do diagrama de estado, que é o de capturar ações (trabalho e atividades que serão executados) e seus resultados em termos das mudanças de estados dos objetos.

O diagrama de atividade mostra o fluxo seqüencial das atividades, é normalmente utilizado para demonstrar as atividades executadas por uma operação específica do sistema. Consistem em estados de ação, que contém a especificação de uma atividade a ser desempenhada por uma operação do sistema. Decisões e condições, como execução paralela, também podem ser mostradas no diagrama de atividade. O diagrama também pode conter especificações de mensagens enviadas e recebidas como partes de ações executadas.

h) Diagrama de componente

O diagrama de componente e o de execução são diagramas que mostram o sistema por um lado funcional, expondo as relações entre seus componentes e a organização de seus módulos durante sua execução.

O diagrama de componente descreve os componentes de *software* e suas dependências entre si, representando a estrutura do código gerado. Os componentes são a implementação na arquitetura física dos conceitos e da funcionalidade definidos na arquitetura lógica (classes, objetos e seus relacionamentos). Eles são tipicamente os arquivos implementados no ambiente de desenvolvimento.

i) Diagrama de execução

O diagrama de execução mostra a arquitetura física do hardware e do *software* no sistema. Pode mostrar os atuais computadores e periféricos, juntamente com as conexões que eles estabelecem entre si e pode mostrar também os tipos de conexões entre esses computadores e periféricos. Especificam–se também os componentes executáveis e objetos que são alocados para mostrar quais unidades de software são executados e em que destes computadores são executados.

O diagrama de execução é composto por componentes, que possuem a mesma simbologia dos componentes do diagrama de componentes, *nodes*, que significam objetos físicos que fazem parte do sistema, podendo ser uma máquina cliente numa LAN, uma máquina servidora, uma impressora, um roteador, entre outros e conexões entre estes

nodes e componentes que juntos compõem toda a arquitetura física do sistema. FOWLER (06) UML(02)

2.1.5 Considerações sobre orientação a objetos

A orientação a objeto é uma estrutura de programação em que um objeto agrupa nele mesmo tanto os dados quanto o código de maneira encapsulada.

O modelo OO focaliza mais o problema. Um programa OO é equivalente a objetos que trocam mensagens entre si. Os objetos do programa equivalem aos objetos da vida real (problema).

A abordagem OO é importante para resolver muitos tipos de problemas através de simulação.

Em termos de linguagens, pode–se esperar uma consolidação, ou uma penetração ainda maior de Java, C++ e *SmallTalk*, que deverão realmente continuar a ser as grandes ferramentas de programação OO.

Além disso, a grande maioria das demais linguagens, ferramentas e ambientes de desenvolvimento já se utilizam forma significante dos recursos da orientação a objeto.

Ferramentas cases importantes disponibilizam recursos para todas as fases de vida dos sistemas e já conseguem gerar código OO diretamente de diagramas corretamente elaborados, como por exemplo as ferramentas da família *Rational*, tendem a contribuir em muito para o crescimento do uso da orientação a objetos.

As linguagens orientadas a objetos apresentam uma nova forma de organizar e estruturar programas e isto leva à necessidade de desenvolver uma nova forma de pensar em resolução computacional de problemas. O paradigma de orientação a objetos conduz naturalmente à implementação de módulos fortemente coesos e fracamente acoplados. Isto torna os programas orientados a objetos mais manuteníveis e suas partes mais reusáveis, facilitando o que em engenharia de software se chama de peça de *software*. Por tudo isso, cada vez mais encontra–se o paradigma de orientação a objetos incorporado nos mais diversos ambientes de computação de uso contemporâneo.

2.2 Middlewares

Os avanços na área de comunicação entre computadores tornaram possível a interligação e o compartilhamento de dados, por todo o mundo. Entretanto, apesar desta

comunicação ser possível a interoperabilidade entre aplicações não é algo relativamente fácil devido à sua intrínseca heterogeneidade e às diferentes características tecnológicas destas aplicações.

Os problemas envolvidos principalmente com a questão da interoperabilidade enfocam quatro aspectos a saber:

1. Interoperabilidade básica – como os programas devem ser escritos para assegurar que o código gerado poderá interagir com código escrito por outros programadores.

2. Atualização de versões – como atualizar um componente do sistema sem requerer que todos os componentes do sistema sejam também revistos e atualizados. O problema em questão é a necessidade do modelo para hoje e para o futuro.

3. Independência de linguagens – garantia de que as aplicações escritas em linguagens diferentes iram conseguir se comunicar.

4. Interoperabilidade entre processos e redes – Necessidade de fornecer aos programadores a flexibilidade e ferramentas para que eles possam escrever as suas aplicações de modo que sejam capazes de operar entre processos e entre plataformas, usando um modelo único de programação.

Várias propostas e aproximações foram feitas para solucionar este tipo de problema. A abordagem deste item pretende analisar separadamente algumas das aproximações, apresentando uma análise comparativa destas tecnologias, de sua interoperabilidade, e de como poderá ser o futuro nesta área.

2.2.1 CORBA – Commom object request broker architecture

A especificação CORBA tem como objetivo, definir uma arquitetura padrão para plataforma comum de entendimento que possa ser adotada por aplicações distribuídas. O seu objetivo principal é permitir a reusabilidade, portabilidade e interoperabilidade entre aplicações orientadas a objetos em ambientes distribuídos heterogêneos, bem como a interligação de múltiplos sistemas de objetos não idênticos.

Esta definição foi proposta pelo OMG pela primeira vez em 1991. Fundada em 1989. A OMG promove a teoria e prática da tecnologia orientada a objetos na indústria de *software*. A força desta organização justifica-se devido ao fato de seus membros

constituírem-se na grande maioria de grandes empresas de desenvolvimento de *software* orientado a objetos.

A estrutura do modelo de referência CORBA contem quatro componentes:

1. Object Request Broker (ORB) permite que objetos façam e recebam pedidos e respostas de forma transparente, num ambiente distribuído.

2. Object Services, uma coleção de serviços (interfaces e objetos) que suportam as funções básicas para usar e implementar objetos. Não determina como é feita a implementação dos objetos numa determinada aplicação. Define convenções para serviços comuns necessários à construção de aplicações distribuídas, embora estes serviços sejam independentes do domínio destas aplicações. Por exemplo, o serviço do ciclo de vida define convenções para criar, copiar, apagar e mover objetos.

3. Commom Facilities, são coleções de serviços que várias aplicações distintas podem partilhar, mas não tão elementares ou básicos como *Object Services*. Por exemplo, um sistema de correio eletrônico pode ser classificado como uma *commom facility.*

4. Aplication Objects, são produtos dum único vendedor, ou aplicação específica desenvolvidas por grupos, que controlam as suas interfaces. Constituem o nível mais elevado do modelo de referência.

O ORB é o núcleo do modelo de referência. É como uma central telefônica: providencia os mecanismos básicos para fazer e receber chamadas (pedidos e respostas), mas não assegura o significado das comunicações entre utilizadores. Também o ORB, por si só, não garante a interoperabilidade ao nível semântico da aplicação. Cabe aos *Object Services* o papel de gerar as interfaces, protocolos e políticas que serão usados pelas aplicações.

A especificação CORBA é baseada no modelo de objetos adotado pela OMG, que define uma semântica comum para especificar, duma forma normalizada e independente, as características externas visíveis dos objetos. Neste modelo clientes requisitam serviços a objetos (também denominados de servidores) através duma interface bem definida. O cliente acessa a um objeto efetuando um pedido ao respectivo objeto. Esse pedido é tratado como um evento que contém informações acerca da operação invocada no

objeto, a sua referência e contexto, lista de exceções que esse pedido pode levantar, e eventuais parâmetros.

O ORB é responsável por localizar o objeto, prepará-lo para receber o pedido, e comunicar-lhe os dados que formam a invocação do pedido ao objeto. Passa então o controle para o objeto, que satisfaz ou não o pedido, e comunica de volta os resultados obtidos ao cliente invocador do pedido.

A definição das interfaces é feita através da Interface Definition Language (IDL), onde é descrito o conjunto de operações que um cliente pode invocar. A IDL suporta o conceito de herança de interfaces como, por exemplo, compiladores de IDL.

A compilação destas interfaces dá origem a bibliotecas estáticas de módulos adaptadores, designados por *Stubs* para objetos clientes, ou *skeletons* de implementação para objetos servidores, que possibilitam o acesso à interface do respectivo objeto durante a execução. A definição dinâmica duma interface de invocação ou de um *skeleton* de implementação é feita acrescentando-a ao serviço de repositório de interfaces ou implementações, respectivamente, onde são armazenadas como objetos que permitem o seu acesso em tempo de execução. Do ponto de vista do objeto, a invocação dinâmica ou a invocação estática através dum adaptador predefinido é transparente, não lhe sendo possível distinguir qual das formas é usada para acessar/fornecer o serviço requerido.

Para que um cliente consiga fazer um pedido a um objeto, tem que comunicar como núcleo ORB, ou através do *Stub* predefinido ou, através da Interface de Inovação Dinâmica. O *Stub* representa o mapeamento entre a linguagem de implementação específica de cada cliente e o núcleo ORB. Assim os clientes podem ser escritos em qualquer linguagem desde que exista o respectivo *Stub*.

2.2.2 COM – Component object model

2.2.2.1 Component software

O problema, do ponto de vista da Microsoft, consiste em criar uma API (*Application Programming Interface*) de serviços do sistema, que seja capaz de operar com múltiplos fornecedores de aplicações duma forma polimórfica. Isto é, um cliente dum serviço poderá usar transparentemente qualquer fornecedor desse serviço, sem possuir algum conhecimento prévio específico acerca de qual sistema ou implementação particular está usando. A abordagem usual é existir um bloco central (mediador) de código que todas as aplicações usam para acessar aos serviços requisitados (caso do CORBA). É trivial

perceber que esta abordagem implica um *overhead* de acesso ao serviço, por vezes demasiado caro às aplicações. Além disso, não existe uma forma simples de aumentar, melhorar, ou adaptar certas capacidades dos serviços aos clientes, sem que se volte a construir e rever esse bloco central de código.

A solução Microsoft consiste num sistema onde programas criem componentes de software que possam ser reutilizados. Um componente de *software* é um pedaço de código em forma binária que pode ser "montado" noutro componente (de qualquer fornecedor) duma forma fácil. Estes componentes terão que aderir a uma certa forma de representação binária externa, mas a sua implementação interna é totalmente livre. Tanto podem ser construídas através de linguagens procedimentais ou através de linguagens orientadas a objetos, se bem que, a orientação a objetos é uma vantagem no mundo de componentes de *software*.

Assim, a programação dos componentes permite uma maior produtividade no projeto, construção, venda e reutilização de *software*.

Para a Microsoft, o COM é o núcleo central de toda uma arquitetura de suporte a sistemas de objetos, que suporta a especificação OLE (*Object Linking and Embebding*) nas suas diversas vertentes.

Para qualquer plataforma (combinação de *hardware* e sistema operacional) o COM define um modo *standard* de representar tabelas de funções virtuais em memória (*viables)*, e um modo *standard* de chamar funções através das *viables*.

Um objeto, na especificação COM, é um pedaço de código compilado que fornece algum serviço ao sistema. Para evitar confusões, chama-se a um objeto COM um objeto componente ou simplesmente, componente.

No COM os objetos, componentes, interagem uns com os outros através de interfaces, que agrupam conjuntos de serviços relacionados entre si. Definem o comportamento e responsabilidades desse componente ao exterior perante determinado contexto. A única forma de acessar um componente é através da interface. Isso obriga a cada um dos componentes ter pelo menos uma interface básica: o *Iunknown*, estipulado pelo *standard* para propósitos específicos. Os dados associados a cada componente nunca podem ser acessados diretamente. Quando um objeto implementa uma interface, esse objeto implementa cada função membro dessa interface e providencia ponteiros para essas funções.

O COM usa GUID's (*Globally Unique Identifiers*) para identificar univocamente qualquer interface (IID) e qualquer classe de componente (CLSID). São também atribuídos nomes, mas estes são locais à máquina. Os GUID's são identificadores universais, isto é, identificam de forma única milhões de componentes através do tempo e do espaço no mundo, que asseguram a correta invocação de determinado componente.

Do ponto de vista do cliente, os objetos são sempre acessados através de ponteiros para interfaces. Um ponteiro, para ter validade, tem de estar dentro do processo (*in-process*), e de fato, qualquer chamada a uma função duma interface chega sempre primeiro a algum pedaço de código residente em um objeto.

Do ponto de vista do servidor, todas as chamadas a uma interface dum objeto são feitas através dum ponteiro para essa interface.

Quando um programador define uma nova interface pode criá-la a partir da linguagem IDL da Microsoft, variante da linguagem *standard* IDL utilizada também pelo CORBA e DCE. O compilador IDL da Microsoft usa essa definição da interface para gerar arquivos tipo *header* para uso de aplicações, código fonte para criar o proxy; e objetos *stub* para lidarem com RPC's.

O *Component Object Library* é o sistema que providencia a mecânica associada ao COM. Possibilita a capacidade de se realizar chamadas ao *Unknown* entre processos; encapsula todo o trabalho de construção associado ao registro de componentes e suas ligações.

2.2.3 Java RMI

O Java *Remote Method Invocation* permite programar objetos distribuídos usando a linguagem de programação orientada a objetos Java. O modelo seguido pela linguagem Java "*Write Once, Run Anywhere*", que possibilita uma mesma aplicação executar em plataformas distintas, desde que execute a *Java Virtual Machine* (JVM), permite incluir o RMI como mecanismo que faculta a distribuição duma aplicação.

Se a linguagem Java abraça a filosofia de "*Write Once, Run Anywhere*", com a RMI esse conceito é estendido para **"***Write Once, Run Everywhere***"**, isto é, através de RMI é possível uma aplicação Java ser facilmente implementada duma forma distribuída. Como o RMI é centrado na linguagem Java, as potencialidades desta linguagem, especialmente a segurança e portabilidade, são implicitamente transportadas para o modelo de computação distribuída.

Ao nível mais básico, o RMI é um mecanismo de igual gênero do *Remote Procedure Call* (RPC), mas específico da linguagem Java. As vantagens do RMI são:

- Orientação a Objetos: fazendo parte da aproximação da linguagem Java, orientada a objetos, RMI beneficia das suas vantagens. Por exemplo, RMI pode passar objetos como argumentos das chamadas e valores de retorno, não estando limitado a tipos de dados pré-definidos. O padrão de desenho duma aplicação mantém-se inalterado, continuando orientado a objetos numa abordagem homogênea.

- Seguro: RMI usa os mecanismos de segurança construídos de raiz na linguagem Java que permite ao sistema ser seguro mesmo através duma rede de dados.

- Fácil de escrever e fácil de usar: RMI torna o processo de transformar uma aplicação em servidora ou cliente tão fácil que bastam três linhas de código para declarar ma aplicação como servidora.

- Conexão a sistema existentes: RMI interage com sistemas existentes através do *Java Native Interface* (JNI), que permite ao código escrito em Java interoperar com aplicações e bibliotecas escritas noutras linguagens de programação como C, C++ ou Assembly. Usando RMI e JNI é possível escrever o código cliente em Java e utilizar a implementação servidora já existente. Similarmente, RMI interage com bases de dados existentes usando *Java DataBase Connectivity* (JDBC) sem modificações do código fonte não-Java.

- *Distributed Garbage Collection*: RMI usa um algoritmo de *Distributed Garbage Collection* para recolher e retirar de memória objetos remotos que não sejam referenciados pelos clientes na rede.

O sistema RMI foi pensado para fornecer um suporte direto e simples à computação distribuída usando a linguagem de programação Java. A sua arquitetura foi desenhada prevendo a expansão e escalabilidade dos servidores tal que o sistema se mantenha coerente.

O sistema RMI consiste em três camadas. A fronteira de cada camada é especificada através duma interface e dum protocolo. Cada camada é independente da

outra e pode ser substituída por implementações e alternativas se conveniente, sem afetar as outras camadas do sistema.

A camada de Referências Remotas é responsável por tratar à semântica da inovação. Por exemplo, é responsável por determinar se o servidor é um único objeto ou é um objeto replicado, necessitando assim gerar conexões múltiplas.

A camada de transporte é responsável pelo estabelecimento e gestão das conexões de comunicação.

Quando um cliente recebe uma referência para um servidor, o RMI faz a tradução das chamadas dessa referência para chamadas ao servidor. O *stub* faz a conversão dos argumentos (*marshling*) do método e depois envia a invocação convertida através da rede.

Do lado do servidor a chamada é recebida pelo sistema RMI e conectado a um *skeleton* que é responsável por traduzir de volta as mensagens e invocar a implementação do método invocado. Quando a execução termina, o skeleton converte o resultado e envia-o ao *Stub* do cliente. O *Stub* reconverte a resposta enviada e, ou retorna o valor do resultado, ou lança uma exceção no caso do método no servidor não tem conseguido executar corretamente. Os *Stub* são gerados quando dá implementação do servidor.

A arquitetura JavaBeans foi pensada para trabalhar bem num ambiente distribuído, que através do Java RMI, quer através da arquitetura CORBA usando Java IDL ou ainda através de JDBC, que permite acesso a base de dados remotas.

O objetivo das *JavaBeans Aplication Programming Interfaces* (API) é definir um modelo de *software* de componentes para a linguagem Java, de tal modo que fabricantes distintos possam reutilizar os componentes desenvolvidos para construir aplicações compostas, usando ferramentas visuais, e executá-las em qualquer plataforma que tenha instalado *Java Virtual Machine*.

Um dos objetivos dos componentes JavaBeans é oferecer um arquivo de componentes independente da plataforma onde executam. Quando um *Bean* é iniciado noutro *Bean*, a funcionalidade é total em todas as plataformas.

O Java RMI possibilita a construção de aplicações distribuídas duma forma sólida e simples. No entanto está fortemente ligada à linguagem Java. Se por um lado isso trás benefícios no que diz respeito à inerente orientação a objetos do sistema ou à sua intrínseca segurança, por outro lado pode constituir uma restrição que limita a liberdade de

escolha da linguagem de desenvolvimento e do uso de aplicações já desenvolvidas noutras linguagens (embora se possam usar métodos e bibliotecas noutras linguagens através do JNI, o seu uso é complicado). Pode-se dizer que o RMI limita-se a estender o conceito de RPC para a linguagem Java, importando assim algumas vantagens e a inerente portabilidade do Java e sua orientação a objetos. Com o conceito JNI as aplicações distribuídas passam a dispor dum paradigma dinâmico de serviços oferecidos e clientes interessados. Basicamente o JNI é um mediador eletrônico para serviços que são dinâmicos, isto é, aparecem e desaparecem, que são implementados numa rede por dispositivos ou *software*.

A API JavaBeans possibilita aos programadores criarem componentes em Java que sejam reutilizáveis e portáteis, em vez das tradicionais soluções monolíticas, lentas de desenvolver e caras. Usadas em conjunção com modelos de distribuição RMI ou CORBA, a sua utilização é potencializada permitindo a distributividade dos *Beans* através da rede. Como no caso do DCOM da Microsoft, esta tecnologia está dependente da vontade dos fabricantes em criarem API's para as suas aplicações proprietárias que suportem JavaBeans. No entanto, quando integrado em aplicações que funcionem com *Java Virtual Machine* (JVM), a sua integração é automática, beneficiando assim da portabilidade da JVM. DEITEL (04) SUN (10)

2.2.4 Estudo Comparativo

A especificação CORBA não define pormenores de implementação, isto tem a ver sobre tudo com os conflitos de interesses entre os membros que compõem o OMG, o que leva a definições intencionalmente vagas e omissas em certas áreas. A falta de pormenores de implementação pode tanto ser uma fraqueza como um motor para a sua generalização, uma vez que proporciona a flexibilidade necessária para implementações específicas. Contudo, a interoperabilidade entre sistemas CORBA disponíveis no mercado, cada vez mais necessária, é assim deixada para o tratamento através do protocolo GIOP (*General Inter-ORB Protocol*) que possibilita a interação entre ORB's distintos.

O DCOM da Microsoft ou Java RMI da Sun Microsystems são executáveis que suportam tecnologia de componentes para aplicação distribuídas. A não ser que se tenha a absoluta certeza de que a aplicação desejada nunca vai ter de ser executada em sistemas que não sejam compatíveis com estas duas tecnologias, é preferível a utilização de tecnologias mais genéricas, e disponíveis em mais plataformas, como o CORBA. Por outro lado, quer da Sun Microsystems, quer da parte da Microsoft existem já produtos que ligam DCOM/CORBA e RMI/CORBA através do GIOP e compromissos no sentido de

assegurarem a continuidade desta interoperabilidade. No entanto, devido às diferentes naturezas das duas tecnologias que tentam interoperar, surgem sempre dificuldades, por vezes bastante difíceis de ultrapassar, o que torna o seu uso impraticável. Por exemplo, CORBA presume um ambiente heterogêneo e multilingüe e por isso deve ter um modelo de objeto neutral, contrastando com o sistema RMI que assume um ambiente homogêneo da JVM, podendo o sistema tirar partido do modelo do objeto da linguagem Java. Outro exemplo, entre o DCOM e o CORBA, é o de herança da implementação. O DCOM não suporta herança de implementação por achar impróprio quando aplicado a modelos de objetos distribuídos, enquanto que o CORBA suporta este tipo de herança.

Na evolução dessas tecnologias uma nova forma de comunicação entre aplicativos está surgindo: os serviços web que possuem a sua estrutura baseada em tecnologias emergentes que serão discutidas na seqüência desse trabalho.

Os middlewares que foram mencionados anteriormente, conforme visto já apresentam uma grande aceitação no mercado e muitas aplicações principalmente que possuem características de plataformas homogêneas rodam muito bem com eles.

Neste sentido a proposta apresentada pela arquitetura orientada a serviços prevê o aproveitamento das soluções que já funcionam de forma satisfatória, apenas adaptando o que for necessário para o uso dos serviços web.

3 SERVIÇOS WEB

Os web services também chamados de serviços web ou web serviços, dependendo da fonte em si, constituem-se na maior promessa de interoperabilidade entre plataformas. Para fins práticos será utilizado o termo serviços web no restante deste trabalho.

No decorrer das pesquisas realizadas foram encontradas inúmeras descrições para o conceito de serviços web. Eis algumas delas:

- Um serviço web é uma aplicação lógica, programável, acessível, que usa os protocolos padrões da internet, para que se torne possível a comunicação transparente de máquina-para-máquina e aplicação-para-aplicação.
- Serviço web é um padrão não proprietário, um formato não binário que por conseqüência possibilita o processamento distribuído em ambientes heterogêneos.
- Serviços web são serviços que podem ser consumidos através da Internet por outras aplicações.
- Serviços web são funções, ou serviços, disponíveis através da Rede Mundial de Computadores. Possuem uma função específica: fornecer um procedimento padrão para interagir com funções e dados. Os serviços web não fazem por si mesmos o que o código de funcionamento faz - executar a lógica dos negócios, processar transações, relatar sobre dados. Ao invés disso fornecem acesso ao código de funcionamento.
- Serviços web permitem que sistemas geograficamente distribuídos conversem entre si, disponibilizando e consumindo serviços.
- Os serviços web apresentam uma nova geração de tecnologia de desenvolvimento. Com ela pode-se criar aplicações modulares e

independentes que são distribuídas facilmente em qualquer estrutura de redes TCP/IP (*Transport Contro Protocol/Internet Protocol)*, pois esse foi um dos princípios fundamentais de sua implementação.

- Serviço web é um software que sabe como se comunicar com outros tipos de software em um ambiente distribuído.
- Serviço web é qualquer tipo de aplicação que possa definir a outras aplicações o que ele faz e pode executar estas funções (serviços) para aplicações autorizadas.
- Serviços web são conjuntos de protocolos e padrões que permitem que aplicações se comuniquem via uma rede (geralmente Internet). Esta comunicação baseada em padrões permite que as aplicações descrevam o que fazem e permite então chamar ou utilizar os serviços de outra aplicação.
- Serviço web é como um software que sabe como falar com outros tipos de software em rede. Um serviço web pode estar próximo de algum tipo de aplicação que tem a habilidade de mostrar para outras aplicações o que faz e qual ação pode executar das aplicações autorizadas.
- Serviços web determinam uma arquitetura para o desenvolvimento de aplicações na Internet, com o conceito de integração “relaxada”. Isto é, os módulos da arquitetura de integração são fracamente acoplados, conferindo assim maior independência entre eles. Esta arquitetura considera também que o cenário ideal para a gerência distribuída de dados é a integração dinâmica, onde serviços são descobertos e acionados quando necessário por ferramentas específicas em registros específicos.
- O serviço web pode ser entendido como uma interface que descreve uma coleção de operações acessíveis em uma rede através de mensagens padronizadas.

Conforme as descrições de serviços web relatadas acima, pode-se perceber claramente que todas elas possuem três aspectos em comum:

- Disponibilizam funcionalidades na web através de um protocolo web padrão. O protocolo mais utilizado é SOAP (*Simple Object Access Protocol*).
- Disponibilizam a descrição das interfaces suportadas de uma forma extremamente detalhada, permitindo assim a construção de aplicações-cliente que consigam comunicar com o serviço web utilizando essas

interfaces. Genericamente, esta descrição é disponibilizada através de um documento XML denominado "*Web Service Description Language*" (WSDL).

- Os serviços web são registrados para permitir que aplicações usuárias as consigam encontrar facilmente. Este registro será feito em *Universal Discovery Description Language* (UDDI).

Para o acesso aos serviços web não é necessário conhecimento sobre a plataforma, modelo de objeto, ou a linguagem de programação que foi usada para implementar o serviço; é preciso apenas solicitar o serviço para que possam receber a resposta da solicitação adequadamente.

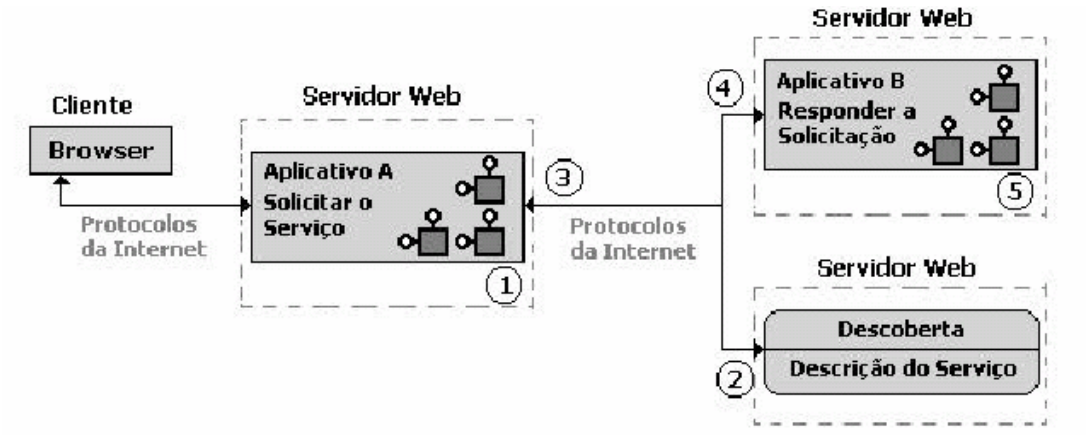

Figura 3.1 Comunicação entre serviços web.

Para que essa solicitação aconteça, no entanto é necessário descobrir, descrever, acessar e interpretar os serviços. A figura 3.1 mostra o uso dos serviços web entre um aplicativo A, que roda no Servidor web - A, e um aplicativo B que roda no Servidor web - B.

O desenvolvedor do aplicativo "A" que deseja utilizar o aplicativo "B" escreve o código do "A" (1) para acessar o serviço do "B" (5). Porém, o serviço do "B" deve ser descoberto primeiro ou conhecido (2) e um contrato (descrição do serviço) tem que existir entre o "A" e o "B". Esse contrato descreve quais serviços o "B" exibe e a semântica utilizada pelo "A" para chamar os serviços do "B". Depois que o desenvolvedor do "A" tiver o contrato e o local do serviço do "B", ele poderá desenvolver uma solicitação (3). O contrato é necessário para que o desenvolvedor saiba como utilizar o serviço. O local dele é relativo tendo em vista que o serviço e o contrato podem existir juntos ou separadamente, ou seja, podem existir no mesmo ou em outro Servidor web. Quando o "B" receber a solicitação (4), ele terá que decodificar a solicitação, em seguida chamar a lógica do serviço e recuperar as

informações na ordem correta e, por fim o "B" deve codificar e transportar para o "A" a resposta da solicitação. WEB (17) W3C (16)

3.1 Histórico

A evolução exibida pelas tecnologias da informação e comunicação pode ser referenciada como uma das mais marcantes ocorridas em segmentos de oferta de serviços na história moderna. Um roteiro que parte do computador central, o *mainframe*, que há alguns anos suportava através de conexões via terminais, o processamento de dados e que nos leva aos atuais sistemas baseados em Internet que apóiam o trabalho de equipes móveis em escala geográfica mundial, pode ser a orientação para que se analise este inigualável processo.

Do gigantesco computador central, *o mainframe*, oriundo dos primeiros experimentos com máquinas processadoras de cálculos do pós-guerra, partiu-se para os computadores de porte médio, que permitiram às organizações a posse de seus próprios sistemas processadores de dados, em virtude de serem operados em menores instalações e de custo inferior às primeiras máquinas comerciais.

Os minicomputadores levaram consigo a semente do que seria a primeira onda de descentralização da tecnologia da informação, demandando que as organizações não mais possuíssem apenas pessoal de preparação de dados - o que até então ocorria - mas de desenvolvimento de soluções. Esta formação levou o mercado a fornecer recursos de hardware e software para suporte deste novo cenário de trabalho.

Daí surgiram demandas por modelos flexíveis de processamento em empresas, que tanto visavam a extrair toda a possibilidade de serviços dos recursos tecnológicos, como permitir a montagem de estruturas de fornecimento de outros novos, apoiados pela tecnologia.

O próximo passo ocorreu com a introdução dos microcomputadores, através do lançamento do revolucionário *personal computer* em 1981, pela IBM. Este fato, inédito na estratégia da maior empresa de informática da época, iniciou uma tendência de descentralização que permitiu a montagem de estruturas ágeis de processamento local de informações por parte de departamentos empresariais, lares e escritórios empresariais. Citações ao computador departamental eram habituais nos anúncios publicitários da época.

Estas unidades foram, em seqüência, integradas na forma de redes de computadores, que visavam o compartilhamento de recursos entre diversas máquinas, bem como de possibilitar a criação de uma arquitetura de processamento de aplicações que permitisse a distribuição do processamento - isto é, cada nó desta nova rede de

computadores interligados ofereceria aos seus usuários o poder de realizar uma parcela importante das tarefas de fluxos e processamento de informações para a aplicação em questão.

Em paralelo, ocorre a popularização o uso da rede Internet, recurso que, na época, ainda era concentrado no meio acadêmico e militar. O surgimento da WWW *(World Wide Web)* e a flexibilização dos órgãos controladores da rede para que se desenvolvesse seu uso comercial são decisivos para tal acontecimento.

Este fato propiciou inovações importantes na arquitetura das aplicações, através da criação de novas alternativas de serviços, sendo a afirmação do modelo cliente-servidor merecedora de destaque. Neste caso, para processamento de uma aplicação distribuída, o usuário de uma rede necessita ter em sua máquina um programa cliente que executará as funções de interface com o usuário e de chamada de serviços, que serão executadas por um servidor com quem se conecta quando necessário.

Este modelo de processamento, hoje encontrado nos serviços de bancos de dados, aplicações de *groupware* e Internet, entre outras, apoiou a criação de uma arquitetura computacional baseada em protocolos de comunicação especiais que possibilitam a implementação de novos serviços, atendendo o que é demandado pelas organizações. Assim a difusão das redes prosseguiu. Com a popularização da Internet nota-se principalmente a introdução organizacional do correio eletrônico estruturado em torno da web.

O comércio eletrônico é considerado um exemplo da nova tendência de dinamização dos negócios entre organizações, e é visto por muitos como a possibilidade do surgimento de uma perspectiva sócio-econômica diferenciada.

O comércio eletrônico e outras aplicações de grandes instituições requerem o uso de soluções distribuídas para a integração de seus softwares. Entretanto um problema que acompanha a evolução da área de tecnologia como um todo precisava ser solucionado: a interoperabilidade. Surgem alguns *middlewaress* importantes, responsáveis por realizar a integração e a interoperabilidade entre os sistemas, entre eles pode-se citar como destaques DCOM, CORBA, RMI. Apesar de apresentarem importantes avanços e possibilitarem a integração de aplicações principalmente em estruturas internas essas tecnologias não conseguiram solucionar como um todo o problema da interoperabilidade.

Na evolução dessas iniciativas surgem novas tecnologias, os serviços web.

Os serviços web são aplicações que comunicam pela Internet, implementadas de forma flexível e fracamente acoplada, estão tendo um papel crescente nas interações entre empresas por meio eletrônico.

No final da década de 90 deu-se início a pesquisas e iniciativas que apontavam para o surgimento desse novo conceito, que num primeiro momento levava apenas a descrição de arquitetura orientada a serviços.

Os pesquisadores advindos principalmente de meios acadêmicos descreviam o que seria a arquitetura orientada a serviços, ou seja, a necessidade de que os serviços precisam compartilhar princípios organizacionais para funcionarem bem em conjunto, de forma flexível e dinâmica. Esta arquitetura focaliza-se em como os serviços são descritos e organizados dinamicamente, propiciando descoberta e uso automáticos.

Para que a descoberta automática seja factível, uma coleção de serviços precisa ser organizada em uma hierarquia de categorias baseadas em o que o serviço faz e como ele pode ser acionado por outro serviço. BURBECK (01)

Segundo pesquisas realizadas entre 1998 a 2000 acreditava-se que as taxonomias dos serviços seriam mantidas e disponibilizadas por serviços de categorização, ou agenciadores (*brokers*), análogos ao que se conhecia para a organização de informações na web, como o Yahoo! ou o *Open Directory* da Netscape, o que hoje se retrata nos registros mantido pela UDDI e eb-XML.

Ainda em pesquisas nesse mesmo período, visando detectar formas e cenários para o comércio eletrônico, apontavam-se 3 cenários para as infra-estruturas de colaboração de serviços B2B:

- Acoplamento rígido, estabelecido em tempo de conexão: A aplicação conhece os detalhes precisos do serviço com o qual vai colaborar porque este foi acoplado durante a concepção. Assim, a aplicação sabe exatamente como interagir com o serviço;
- Acoplamento dinâmico a colaborador pré-estabelecido: A aplicação sabe como solicitar a um agenciador por um serviço específico, isto porque o programador codificou uma consulta específica a ser feita ao agenciador. Entretanto detalhes da interação dependem da descrição do serviço retornada pelo agenciador em tempo de execução;
- Acoplamento dinâmico e escolha dinâmica do colaborador: A aplicação sabe a semântica e as chamadas da API (*Application Program Interface)* do serviço a ser usado, no entanto consulta um agenciador com um padrão de busca que permite o retorno de uma série de alternativas. A aplicação escolhe então um serviço da lista em tempo de execução.

Enquanto no modelo cliente-servidor, um servidor é uma entidade única esperando para realizar uma tarefa, um serviço web distingue a tarefa a ser executada da entidade (ou entidades) que a executará. Pode haver várias entidades, cooperando pró-ativamente para atingir um objetivo único. Tem-se um modelo pró-ativo de aplicações, onde um serviço tenta conexão com outros serviços, procurando-os através de diretórios de registros, fazendo chamadas e interagindo dinamicamente com múltiplos parceiros a fim de construir a lógica da aplicação.

Nos estágios iniciais do comércio eletrônico as aplicações estavam focalizadas nos dois primeiros cenários. Os serviços web, por sua vez estão focalizados no terceiro cenário que é o mais flexível e adaptativo. BURBECK (01)

Alguns autores retrataram essa nova realidade, como BURBECK (2000)

> "... os sistemas interativos de computadores, que até então limitavam-se às redes locais, tornaram-se operacionais em redes remotas, e o paradigma computacional passou da mera conexão entre computadores à computação cooperativa, independentemente da localização dos parceiros interagentes."

Em outro texto exemplifica de forma clara:

> "... quando do desenvolvimento das linguagens de terceira geração a invenção da sub-rotina (ou função) permitiu aos programas serem subdivididos em unidades funcionais. A partir dessa inovação técnica, surgiram as metodologias de concepção e análise que se utilizaram da decomposição funcional, melhorando a modularidade e facilitando o processo de concepção de sistemas cada vez mais complexos. A noção de objeto que combina funções e dados em uma mesma unidade encapsulada, introduziu novos construtos organizacionais técnicos, tais como as noções de classe, herança e polimorfismo. Estas inovações deram origem as novas metodologias OO, técnicas de análise e práticas de concepção. No início (fim dos anos 80), eram consideradas pelos mais céticos mais como práticas de decomposição funcional ligeiramente repensadas e, sobretudo, renomeadas (os programas viram objetos, as chamadas de procedimento viraram mensagens, as funções viraram métodos). Posteriormente com a prática e a evolução das ferramentas de concepção tornou-se claro que juntos os novos princípios representavam algo fundamentalmente diferente. A vantagem principal foi promover uma maior reutilização de código (bibliotecas de classes de objetos). Assim, as classes são organizadas em hierarquias e ferramentas permitem navegar de forma que as bibliotecas de classes tornam-se facilmente acessíveis para programadores e analistas."

A partir destas colocações, percebe-se que os serviços web representam uma evolução, uma nova forma de processamento computacional que requer novos princípios

técnicos de organização. Cada serviço individual, diferentemente de funções e objetos, é concebido para satisfazer as regras de negócios de uma empresa enquanto colaboradora com aplicações ou serviços de outras organizações.

Neste novo modelo orientado à serviços, a habilidade de descrevê-los tem papel central para o sucesso de sua publicação para uso, sua categorização precisa, e suas possibilidades de busca correta para a satisfação de uma necessidade. Assim, as descrições dos serviços constituem o meio lógico pelo qual se dá a troca de informações que possibilitam a publicação, a descoberta e o acoplamento entre serviços. BURBECK (01)

Em termos históricos, houve uma evolução das aplicações monolíticas em mainframes, para o modelo tipo cliente-servidor das duas últimas décadas (distribuído localmente em redes locais) impulsionado pela democratização das LANs. Agora inicia-se a compreensão de um novo modelo com características próprias, globalmente distribuído e orientado à serviços, os serviços web.

Um fator decisivo para a consolidação deste novo conceito como um segmento da indústria de tecnologia foi a formação de um consórcio, designado como *web services Interoperability Organization* (WS-I), objetivando o desenvolvimento de padrões. Tal grupo reuniu grandes líderes do mercado de tecnologia, tais como IBM, Microsoft, SAP, Oracle, BEA System, Fujitsu entre outras, endossado por duas importantes empresas de consultoria nesta área: o Gartner Group e a Forrester Research. A reunião destes grandes competidores foi marcada, espantosamente, pelo interesse genuíno e autêntico de se estabelecer padrões satisfatórios para todos os envolvidos. Adiciona-se a isto o fato de que este consórcio, mais nitidamente através da Microsoft, vem trabalhando junto ao W3C (www.w3.org), entidade responsável pela padronização de diversas linguagens e protocolos de Internet, donde resultou a padronização do SOAP (*Simple Object Access Protocol*) e de outros padrões que dão suporte ao uso de serviços web. W3C (16) WEB (17)

3.2 A Arquitetura Orientada a Serviços

A arquitetura orientada a serviços destaca dois elementos importantes em qualquer sistema papéis e operações. Por papéis, entende-se os diferentes tipos de entidades; as operações representam as funções executadas por essas entidades, para que o serviço web possa funcionar. A figura 3.2 apresenta o modelo de serviços web através de um diagrama.

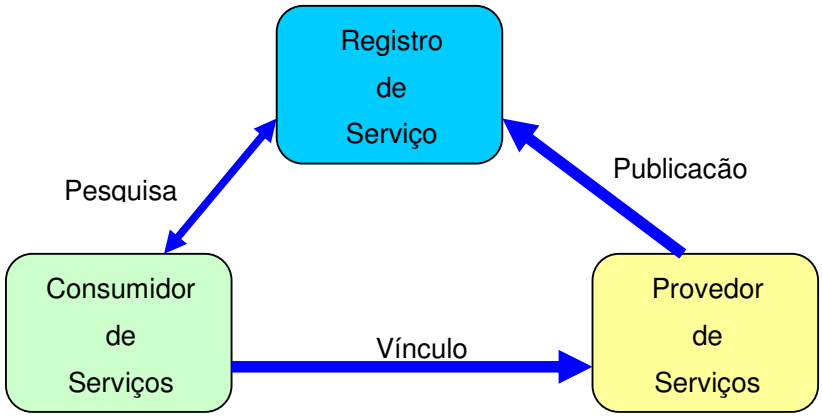

Figura 3.2 Papéis e interações da arquitetura de serviços web

Aqui é possível identificar três tipos de papéis em um ambiente de serviços web típico, bem como as operações que eles executam para fazer os serviços web funcionarem. Os papéis mostrados no diagrama são:

- **O Provedor de serviços** - O provedor de serviços é a entidade que cria o serviço web. Também chamado de *service provider*, ou fornecedor, o provedor de serviços precisa fazer duas coisas para ter acesso a todo o potencial de um serviço web. Primeiramente, precisa descrever o serviço web em um formato padrão, que seja compreensível por qualquer empresa que possa usar esse serviço web. Em segundo lugar, para alcançar um grande público, o provedor de serviços deve publicar os detalhes sobre seu serviço web em um registro central que esteja publicamente disponível para todos os interessados.

- **O Consumidor de serviços** - Qualquer empresa que utilize um serviço web criado por um provedor de serviços é conhecido por consumidor de serviços, usuário ou ainda *service requester*. O consumidor de serviços pode conhecer a funcionalidade de um serviço web a partir da descrição disponibilizada pelo provedor de serviços. Para recuperar os detalhes o consumidor de serviços realiza uma pesquisa sobre o registro onde o provedor de serviços publicou sua descrição do serviço web. O mais importante é que o consumidor de serviços pode obter, a partir da descrição do serviço, o mecanismo para vínculo com o serviço web do provedor de serviços, podendo então acessar esse serviço.

- **O Registro dos serviços** -Um registro de serviços é a localização central onde o provedor de serviços pode relacionar seus serviços web, e no qual um consumidor de serviços pode pesquisar sobre os serviços web

registrados. Os provedores de serviços, também conhecidos como *service broker* normalmente publicam sua capacidade de serviços web no registro do serviço para que, dessa forma, os consumidores do serviço os encontrem e, em seguida, vinculem-se ao seu serviço web. Tipicamente, informações como detalhes da empresa, serviços por ela fornecidos e detalhes sobre cada serviço, inclusive detalhes técnicos, são armazenadas na descrição do serviço.

No diagrama apresentado na figura 3.2 é possível identificar ainda três operações que são fundamentais para o funcionamento dos serviços web, quais sejam: pesquisa, vínculo e publicação. É necessário obter uma comunicação entre as aplicações, sem considerar o tipo de linguagem na qual a aplicação foi escrita, a plataforma onde a aplicação está sendo executada e outras considerações. Para que isso ocorra, precisa-se do uso de padrões em cada uma das operações envolvidas:

- **Descrição:** A maneira padronizada para descrever os serviços web, a Web *Service Description Language* (WSDL) é um padrão que utiliza o formato XML. Basicamente, o documento WSDL define os métodos que estão presentes no serviço web, bem como os parâmetros de entrada/saída para cada um dos métodos, os tipos de dados, o protocolo de transporte usado e a URL da extremidade onde o serviço web será hospedado.
- **Publicação ou Registro:** O padrão *Universal Description, Discover, and Integration* (UDDI) permite que os provedores de serviços publiquem detalhes sobre suas empresas e os serviços web fornecidos em um registro central. Esta é a parte relativa à descrição (*Description*) do UDDI. Também fornece um padrão para permitir que os consumidores de serviços localizem provedores de serviços e detalhes sobre seus serviços web. Esta é a parte relativa à descoberta *(Discovery*) do UDDI.
- **Vínculo:** O Simple Object Access Protocol (SOAP) é um mecanismo superficial do XML, usado para trocar informações entre aplicações, independentemente do sistema operacional, da linguagem de programação ou do modelo do objeto.

3.3 Camadas Conceituas da SOA

Para que as operações fundamentais dos serviços web: publicação, localização e vínculo sejam implementadas é necessário uma pilha básica de serviços que será apresentada em cada aplicação que se destine ao desempenho de um determinado papel no sistema de serviços web.

Essa pilha básica de serviços web é apresentada na figura 3.3.

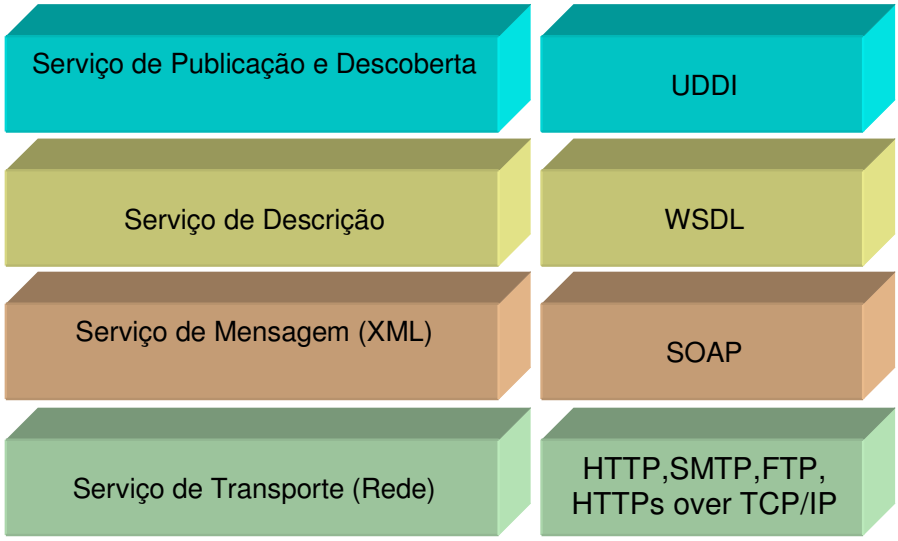

Figura 3.3 Camadas conceituais da SOA

Os blocos apresentados a esquerda na figura 3.3 são camadas conceituais, enquanto os rótulos mostrados à direita de cada um dos blocos em camadas correspondem às tecnologias reais que estão presentes em cada camada.

3.3.1 Serviço de transporte

Essa camada da pilha de serviços web é responsável pela disponibilização dos serviços web, tornando-os acessíveis por intermédio de algum dos protocolos de transporte disponíveis, como HTTP, SMTP, FTP e outros. Os serviços web são construídos com base em padrões de comunicação existentes, que os torna independentes do transporte. No cenário atual, o HTTP é o protocolo de comunicação mais amplamente utilizado. WEB (17) W3C (16)

3.3.2 Serviço de mensagens

A próxima camada na pilha básica de serviços web é a troca de mensagens. Essa camada define o formato de mensagem usado na comunicação entre aplicações. O padrão usado com regularidade pelos serviços web é o SOAP, um protocolo com base em

XML para a troca de informações entre aplicações, independentemente do sistema operacional, do ambiente de programação e do modelo do objeto.

A XML é uma linguagem de marcação, aberta, utilizada para troca de dados principalmente na Internet. Maiores informações sobre essa tecnologia que esta revolucionando as relações na web são apresentadas no capítulo 4.

O SOAP é um protocolo superficial e simples, pois tudo o que é necessário é que os dados de XML passem pelo HTTP. Ele não tenta definir um modelo de programação ou implementar APIs específicas. A especificação do SOAP é tratada com maiores detalhes no capítulo 5 desta dissertação. WEB (17) W3C (16)

3.3.3 Descrição do serviço

A camada de descrição do serviço fornece um mecanismo ao provedor de serviços, a fim de descrever a funcionalidade proporcionada pelo serviço web. Um documento WSDL descreve um serviço web como uma coleção de extremidades ou portas operando independentemente, tanto nas mensagens orientadas a documento, quanto nas orientadas a procedimentos. A WSDL é para um serviço web o que o CORBA IDL é para o CORBA ou a Microsoft MIDL é para o os componentes COM, e assim por diante.

A WSDL apresenta a definição de um serviço web em duas partes. A primeira representa uma definição abstrata independente do protocolo de transporte de alto-nível, de um serviço web, enquanto a segunda representa uma descrição de ligação específica para o transporte na rede.

Em um alto nível de abstração, um serviço web contém os seguintes elementos:

- **Tipos de porta** - Os elementos *portType* contêm um conjunto de Operações representadas como elementos *operation*, que estão presentes em um serviço web, como, por exemplo, *GetStockQuote, SellStock, BuyStock* e outros. É análogo a uma interface em Java. Uma operação pode, portanto, ter uma mensagem de entrada e uma mensagem de saída. Além disso, ela pode ter apenas uma mensagem de entrada ou saída, da mesma maneira que uma chamada de método normal.
- **Mensagens** - Um elemento *message* contém uma definição dos dados a serem transmitidos. É semelhante ao parâmetro na chamada do método.
- **Tipos** - O elemento *types* contém os tipos de dados que estão presentes na mensagem.

- **Vínculos** - O elemento *binding* mapeia os elementos *operation* em um elemento *portType*, para um protocolo específico.

A descrição da WSDL de um serviço web contém uma coleção de portas – cada porta se associa a uma extremidade na qual o serviço web pode aceitar comunicações (como um URL ou um endereço de e-mail) com um elemento *binding* particular, o que descreve as mensagens aceitas para esta porta. Maiores detalhes sobre a tecnologia WSDL são descritos no capítulo 6. WEB (17) W3C (16)

3.3.4 Publicação e descoberta do serviço

A partir de um documento WSDL, um consumidor de serviço pode determinar os detalhes do serviço web, como as diferentes operações, tipos de dados, extremidades, protocolos de vínculo e outros. Para tanto é necessário que essa definição seja publicada.

Em vez de publicar o documento WSDL para cada possível cliente, um provedor de serviços estará bem servido se publicar as informações sobre seu serviço web em um registro central que esteja disponível publicamente para os consumidores de serviço interessados, publicando conjuntamente informações relacionadas ao negócio, como o nome dos seus negócios, os diferentes serviços web por eles oferecidos, entre outros.

Da mesma maneira, os consumidores de serviço podem querer encontrar os diferentes serviços web que sejam disponibilizados pelos provedores de serviço. Eles podem querer avaliá-los independentemente, antes de integrar esses serviços às suas aplicações. Conseqüentemente, o ideal seria pesquisar no mesmo registro central, onde os provedores de serviço já publicaram seus detalhes. Os consumidores de serviço também podem realizar uma pesquisa de nível mais amplo, com base em categorias comerciais, para encontrar as empresas nessas categorias e, a seguir, fazer o *drill down* em detalhes adicionais sobre os serviços web por elas oferecidos.

Existem muitas possibilidades além das apresentadas acima. Entretanto, o mais importante é a necessidade de um registro central, onde os provedores de serviços e os consumidores de serviços trabalham em conjunto para publicar e recuperar as informações apropriadas. Em outras palavras, é necessária uma especificação para a publicação e descoberta dos serviços oferecidos.

A especificação para a publicação e localização de informações comerciais é a especificação *Universal Discovery, Description, and Integration* (UDDI). A especificação UDDI fornece uma estrutura comercial, usada para descrever um determinado negócio. A estrutura de negócios contém as seguintes informações:

- **Negócio** - Contém as informações comerciais, como o nome do negócio, o contato comercial, e outras. Contém uma ou mais instâncias da estrutura do serviço. A especificação UDDI também permite que uma definição comercial contenha um código de classificação que a identificará como pertencente a uma determinada categoria de negócios, como, por exemplo, instituições de empréstimo financeiras.
- **Serviço** - Uma estrutura de serviço captura os diferentes serviços web fornecidos por este negócio. Cada serviço web contém uma ou mais estruturas de especificação técnica. Uma empresa pode ter, por exemplo, dois serviços web; um serviço web representando o status do pedido de vendas e um representando o status do estoque. A especificação UDDI também permite que uma definição de serviço contenha um código de classificação que a identificará como pertencente a uma determinada categoria de serviços, como, por exemplo, a de verificação de crédito.
- **Especificação Técnica** - As especificações técnicas contêm detalhes técnicos sobre um serviço web. Uma das especificações técnicas é a especificação WSDL, à qual um consumidor de serviços pode fazer referência, e determinar os detalhes técnicos sobre o chamamento do serviço web.

A UDDI é apresentada em detalhes no capítulo 7.

Após a análise dos papéis, operações e tecnologias envolvidas com a arquitetura de serviços é possível visualizar as relações de uma forma mais completa, conforme exposto na figura 3.4.

A primeira etapa do provedor de serviços envolve a codificação do arquivo WSDL. Existem diversas ferramentas disponíveis no mercado, atualmente, que auxiliam na geração do arquivo WSDL, a partir de definições do objeto existente. A seguir, é necessária a publicação de informações sobre si mesmo, que representem o negócio e a especificação técnica do serviço web, como um arquivo WSDL, no registro UDDI central. Assim, a descrição do serviço web por meio da codificação do arquivo WSDL é capaz de capturar a camada Descrição do Serviço, enquanto a publicação de informações comerciais e o arquivo WSDL representam a camada Publicação do Serviço vista anteriormente na pilha de serviços web.

A aplicação do consumidor do serviço pode descobrir os serviços web que ele esteja interessado em usar. A descoberta envolve não apenas a pesquisa de negócios e seus serviços, como também o carregamento das especificações técnicas mencionadas no

arquivo WSDL. Essa etapa da descoberta corresponde à camada da Descoberta do Serviço na pilha básica de serviços web.

Por último, a aplicação do consumidor do serviço utiliza o arquivo WSDL para determinar as mensagens que precisam ser passadas na comunicação com o serviço web do provedor de serviços. Determina, também, as informações sobre o vínculo, que é o SOAP pelo HTTP. A seguir, é feito o envio através de solicitações de SOAP e ocorre o recebimento das respostas de SOAP apropriadas. Esta etapa corresponde à camada de Serviços de Mensagens e Transporte na pilha de serviços web. WEB (17) W3C (16)

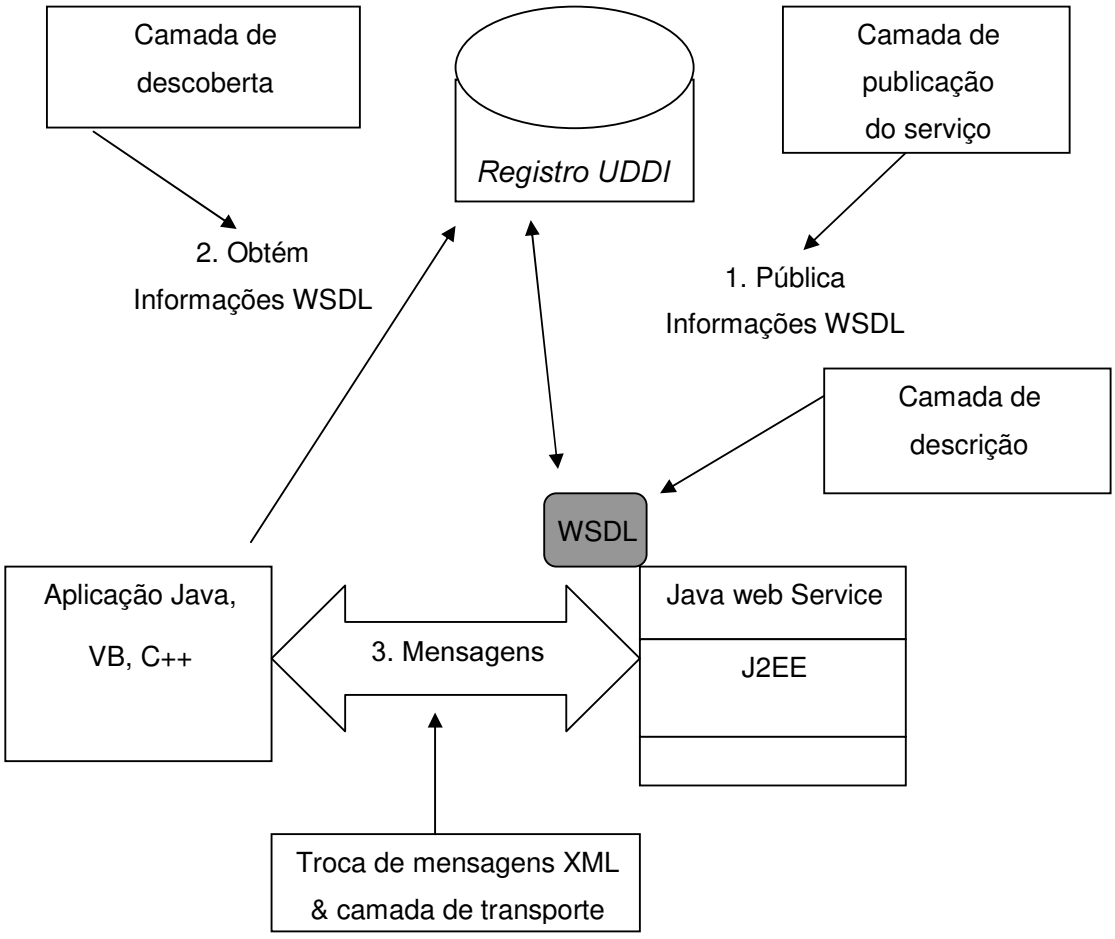

Figura 3.4 Estrutura funcionamento SOA

3.4 Tecnologias Envolvidas

Para implementar um serviço web, a W3C define quatro padrões primários, que já foram enfocados anteriormente, quais sejam:

1. XML (eXtensible Markup Language) - XML é uma linguagem padrão que especifica um formato de dados no qual todos os aplicativos que entendem XML devem concordar. Sem XML, a noção de serviços web não seria possível. XML é a linguagem dos serviços web. Ela é equivalente ao ASCII para a Internet, onde dados ou conteúdo que são expressos em XML devem ser entendidos por qualquer aplicativo que entende XML;
2. SOAP (*Simplified Object Access Protocol*) - SOAP é um protocolo utilizado para chamar procedimentos remotos pela internet. Ele é um documento XML que define, entre outras coisas, como uma aplicação pode dizer a um servidor que determinado objeto deve ser carregado, qual método deve ser executado, com quais parâmetros e qual é o valor de retorno. O necessário, para se manipular SOAP é um Parser XML e um Servidor web que rode HTTP.
3. WSDL (*Web Services Description Language*) - WSDL é um documento XML. Serve de base para a descrição dos serviços. Proporcionam aos clientes de um serviço web todos os detalhes necessários que precisam para conectar e usar o serviço web.
4. UDDI é o diretório para serviços web. Para transformar a atual web semi-estática numa web dinâmica, dotada de serviços colaborativos, é necessário criar um diretório mundial para consultas dos serviços disponíveis, que as empresas divulgarão e procurarão serviços de forma dinâmica. Tal como uma lista telefônica, o UDDI contém páginas brancas e amarelas que permitem procurar os serviços em função do nome da empresa ou do tipo de atividade que exercem. Porém, o UDDI oferece, também, páginas verdes, e indicam como negociar com as empresas que figuram no diretório, designando, por exemplo, os seus procedimentos comerciais ou descrevendo os serviços que disponibilizam. Brancas, amarelas ou verdes, as páginas UDDI possuem um modelo de dados baseado em XML, e o seu acesso exige a utilização do SOAP, tanto para as preencher como para pesquisar os serviços que as contêm.

De uma forma mais simplista a figura 3.5 apresenta a relação dessas quatro tecnologias. Esse diagrama se assemelha aos que foram vistos anteriormente, enfocando apenas de maneira mais clara as tecnologias envolvidas.

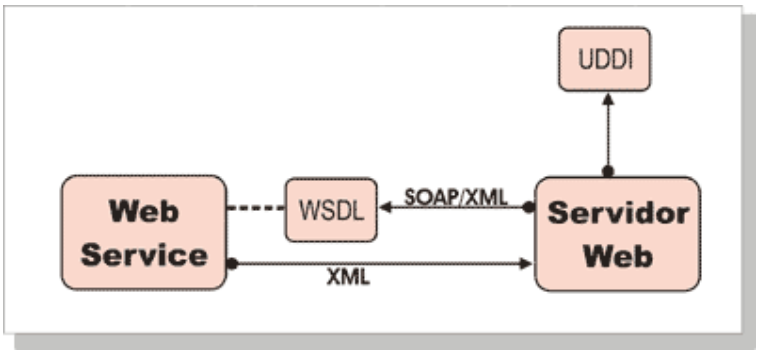

Figura 3.5 Tecnologias utilizadas no serviço web

Na figura 3.6 é possível entender melhor a uso dos recursos possibilitados pelos serviços web e a existência real da interoperabilidade.

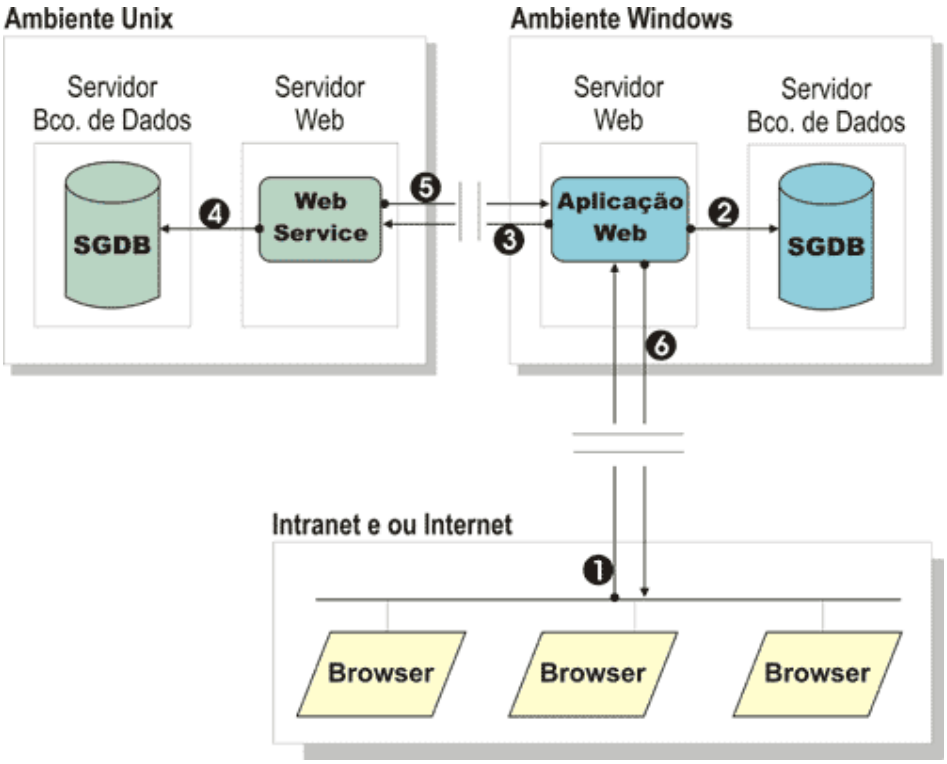

Figura 3.6 Interoperabilidade entre os serviços web

O fluxo da figura representa na realidade o ciclo de uma aplicação, qual seja:

1) Cliente solicita serviço da plataforma ativa ao servidor web.
2) Servidor web entra em processo, acessando base de dados local para obter parte das informações a serem devolvidas ao Browser.

3) Aplicação web faz acesso a um serviço web, que é executado em um servidor de plataforma desconhecida. Neste momento o contexto da aplicação inclui o cliente(*browser*), o servidor web e o servidor do serviço web.

4) O aplicativo serviço web faz acesso a base de dados residente em seu local para obter informação.

5) Serviço web devolve a informação ao aplicativo servidor web solicitante.

6) Aplicação Servidor web, mescla as informações obtidas localmente (passo 2), com as informações obtidas do serviço web (passo 5), retornado-os ao Browser solicitante.

Assim é encerrado todo ciclo do aplicativo.

Essa interface facilita o desenvolvimento de sistemas orientado a componentes, adequado para a reutilização de serviços.

Estratégias de integração baseadas em serviços web oferecem vários benefícios. Pode-se destacar a flexibilidade para alterar implementações de serviços ou adicionar novos recursos ao ambiente distribuído. Além disso, estas estratégias possuem simplicidade de representação, pois são encapsulados detalhes referentes à implementação dos serviços.

Pelo que foi apresentado no decorrer desse capítulo, até o presente momento pode-se concluir que um XML Web Service é um serviço de software publicado na web através do SOAP, descrito com um arquivo WSDL e registrado em UDDI.

3.5 Vantagens

Como a interface do serviço web é definida exclusivamente em termos de mensagens que o serviço aceita e gera, os consumidores do serviço podem ser implementados em qualquer plataforma e linguagem de programação, desde que criem e aceitem as mensagens definidas pela interface do serviço web.

A diferença dessa implementação com o CORBA ou com o DCE é que o SOAP é significantemente menos complexo que as abordagens anteriores, assim a barreira da mudança para uma implementação SOAP é menor. Hoje existem implementações de grandes companhias de software, mas também existem muitas implementações construídas e mantidas por um único desenvolvedor. Outra diferença significativa é trabalhar com protocolos padrões de web - XML, HTTP e TCP/IP. Um grande número de companhias já possui uma infra-estrutura web, pessoas com conhecimento e experiência na sua

manutenção, e novamente, o custo para uma mudança para o XML web services é menor que para as propostas anteriores.

3.6 Aplicações

Os serviços web mais simples são fontes de informação, que podem ser facilmente incorporados às aplicações - preços de produtos, cotas de estoque, previsões de tempo e temperatura, resultados de partidas esportivas.

É fácil imaginar uma grande classe de aplicações que podem ser feitas para analisar e agregar informações importantes, e apresentá-las de várias formas; por exemplo, uma planilha que resume sua situação financeira - títulos, contas bancárias, empréstimos e outros. Se essa informação estiver disponível através de serviços web, a planilha pode ser atualizada continuamente.

Algumas dessas informações podem ser abertas ou podem requerer uma autorização de acesso para o serviço. Muitas dessas informações já estão disponíveis na web, mas os serviços web tornam o acesso programático mais fácil e confiável.

A publicação de aplicações existentes através de serviços web permitirá a construção de aplicações novas e poderosas. Por exemplo, pode-se desenvolver uma aplicação de aquisições que automaticamente obtém os preços de vários fornecedores, possibilitando que o usuário selecione um fornecedor, faça o pedido e acompanhe o carregamento até o recebimento. A aplicação do fornecedor pode também utilizar serviços web para checar o crédito do cliente, cobrar da conta e solicitar o transporte da carga para uma transportadora.

Serviços web têm sua utilidade em várias aplicações. Podem ser utilizados para conectar sistemas da empresa com serviços de parceiros, prover um serviço em tempo real ao consumidor de um modo melhorado, cadeias de fornecimento altamente otimizadas ou canais de entrega mais amplos. Internamente, os serviços web podem ligar os serviços de uma companhia para otimizar os processos de negócio, disponibilizar conexões em temo real a funcionalidades de terceiros. Podem ser utilizados ainda para agregar conteúdo interno e externo a aplicações para prover portais mais dinâmicos e colaborativos.

Um outro bom exemplo para o uso de serviços web seria a cobrança bancaria. Hoje um título de cobrança só pode ser pago em qualquer banco até a sua data de vencimento, após isso somente no banco que deu origem ao titulo junto à empresa. Este problema existia em virtude de não se ter até então, uma forma dos bancos se comunicarem de maneira rápida e segura, com regras comuns. Sugestões estão sendo apresentadas para

solucionar estes problemas e, a novas proposições para transição de documentos bancários, com certeza contemplam o uso do protocolo SOAP. WEB (17) W3C (16)

3.7 Tipos de Serviços Web

Os serviços web podem ser de dois tipos: simples e complexos.

- **Serviços web simples** são caracterizados pelos padrões SOAP, UDDI e WSDL, que juntos provêm um mecanismo de "requisição e resposta". Por natureza, não são transacionais, mas tem uma vasta gama de aplicações, como notícias, consultas, previsões do tempo e outras operações simples.
- **Serviços web complexos,** também chamados de serviços web de negócios, são os que requerem regras de negócios mais elaboradas, ou seja, envolvem transações. Uma venda ou uma reserva de viagem pode exemplificá-lo. Para prover tal sofisticação, utilizam-se de *standards* como o eb-XML e o RosettaNet. Na verdade, tais padrões são especificações mais elaboradas utilizando-se das mesmas tecnologias dos serviços simples: SOAP, WSDL e UDDI.

A maioria das plataformas/ferramentas de serviços web, disponíveis no mercado, atualmente, é capaz de representar serviços web simples. Os serviços web complexos, por outro lado, são representações dos cenários de e-business do mundo real, onde existe a colaboração e um envolvimento entre vários negociantes. Uma forma, de alto nível, para estabelecer a diferença entre serviços web simples e complexos consiste em considerar a analogia entre os servidores web e os servidores de aplicação.

3.7.1 Serviços web simples

Os serviços web simples são os serviços predominantes atualmente. Pode-se examinar sites diversos de serviços web simples, como http://www.salcentral.com ou http://www.xmethods.com, que apresentam um diretório listando centenas de serviços web que se pode chamar, atualmente, a partir do programa de cliente. O maior número deles corresponde ao tipo solicitação-resposta simples de serviços web e contém informações, na maioria das vezes. Um mecanismo típico para implementar serviço web simples consiste em examinar os sistemas existentes na empresa e definir se determinada funcionalidade do seu sistema pode ser exposta como um serviço web. Eis alguns exemplos:

- Uma empresa de validação de cartão de crédito está interessada em apresentar sua funcionalidade já existente de verificação de cartões, como

um serviço web com base em SOAP. Comerciantes on-line são assinantes em potencial deste serviço web.

- Uma agência de empregos deseja publicar suas mais recentes ofertas de emprego como um serviço web com base em SOAP. As empresas interessadas em contratações são assinantes em potencial deste serviço web.
- Um sistema de reservas de passagens aéreas deseja publicar suas tarifas aéreas recentes como um serviço web. Agências de viagens são assinantes em potencial deste serviço web.

3.7.1.1 Componentes centrais do serviço web simples

Para atender as necessidades de serviços web simples é preciso determinar quais tipos de componentes centrais são necessários, quais sejam:

- **Mecanismo de análise do XML**: Caso as solicitações do SOAP estejam em XML, precisa-se de um mecanismo de análise de XML, para analisar a solicitação do SOAP de entrada no modelo do objeto, no nome da operação, nos parâmetros apropriados, e assim por diante. O mecanismo de análise do XML também pode funcionar como um mecanismo de validação do XML para garantir que o XML atenda a especificação SOAP.
- **Mecanismo tradutor do XML:** A resposta recebida dos objetos nem sempre estão em uma forma que atenda a todos os clientes. Por exemplo, pode-se ter diferentes tipos de dispositivos que chamem o serviço web e que provavelmente seja necessário traduzir o XML em diferentes formas, antes de transmiti-lo de volta ao cliente. O mecanismo tradutor do XML atinge este objetivo.
- **Segurança:** Precisa-se agora de uma autenticação básica e de serviços de autorização para verificar a validade das credenciais do usuário, chamando o serviço web. Esta é a função do componente de segurança.
- **Mecanismo de logging:** Finalmente, deve-se ter um serviço de *logging* para registrar tudo o que ocorrer durante a chamada de chamamento. Um mecanismo de *logging* eficiente e bem projetado auxilia não apenas durante a depuração, como também no caso de auditoria e faturamento.

Além dos componentes centrais destacados acima, também precisa-se de uma camada inferior no sistema de serviços web, que seja capaz de fornecer serviços de sistema

que manejem protocolos de comunicação como RMI, IIOP, e outros. Os serviços do sistema não são expostos diretamente ao do usuário, é mais provável que sejam construídos no sistema.

3.7.2 Serviços web complexos

Os serviços web complexos, ou como já citado, os serviços web de negócios resolvem os problemas de *e-business* do mundo real. Os serviços web complexos são capazes de aceitar não apenas o tipo solicitação-resposta simples de serviços, como também processos comerciais de execução extremamente colaborativos. Esses negócios não funcionam isoladamente. Normalmente, um processo de negócio normal envolve várias conversações entre parceiros comerciais, a troca de dados entre vários parceiros comerciais, e assim por diante. Freqüentemente esses processos têm curta duração, mas há casos em que podem durar vários dias.

Para que os serviços web capturem esses processos comerciais, existem várias áreas que devem ser abordadas.

O serviço web simples não resolve problemas de *e-business* relacionados ao mundo real. Alguns desses problemas, que têm participação significativa em todas as transações comerciais, são a segurança, a qualidade do serviço, o gerenciamento da transação, contexto/privativo, e outros. WEB (17) W3C(16)

3.7.2.1 Segurança

A segurança é um dos problemas mais importantes a serem considerados, sempre que for necessário realizar alguma transação comercial. Sempre que for apresentada uma nova tecnologia a uma empresa, uma das áreas de maior interesse será a segurança. Sem um sólido modelo de segurança na aplicação ou tecnologia, ela não será aceita por nenhum negócio destinado a realizar negócios na Internet. O mesmo se aplica a serviços web.

A segurança normalmente inclui as três áreas a seguir:

- **Privacidade**: É preciso garantir que a conversação entre o cliente e o serviço web seja criptografada, de maneira que ninguém possa espionar a conversação.
- **Autenticação**: É importante que tanto o cliente quanto o serviço web confirmem a identidade um do outro.

- **Não-repúdio**: É importante que o serviço web mantenha um registro com os chamamentos, de maneira que não será seja possível negar o envio da mensagem.

Os serviços web podem abranger a segurança em dois níveis: segurança no transporte e segurança na mensagem**.**

A segurança deve ser abordada em cada nível da pilha dos serviços web. Alguns dos padrões que estão sendo desenvolvidos atualmente, para resolver problemas relacionados à segurança nos serviços web são:

- **XKMS** - O padrão XML Key Management Services (XKMS) fornece um padrão para distribuir e gerenciar chaves para a comunicação segura entre extremidades.
- **XML Encryption** - Este padrão ajuda a criptografar a mensagem do remetente, e decriptografá-la no destinatário final, o que torna a mensagem ilegível por outros elementos durante o transporte.
- **SAML**- Security Access Markup Language (SAML) nos fornece um mecanismo para determinar direitos de acesso às extremidades.

Maiores especificações sobre pesquisas na área de segurança para serviços web, envolvendo os protocolos relacionados são descritos no capítulo 8.

3.7.2.2 Mensagens confiáveis

As mensagens confiáveis são um importante componente de qualquer sistema de *e-business*, permitindo garantir que a mensagem foi entregue ao destinatário planejado. As situações onde a mensagem não é entregue, devido a algum problema, também são tratadas. Dependendo dos recursos apresentados, tem-se o reenvio, de acordo com a necessidade.

As mensagens confiáveis devem ser construídas tanto na camada do serviço de transporte quanto na camada de serviço de mensagem da pilha de serviços web. Elas precisam resolver problemas como a solicitação de mensagens, as restrições quanto ao tempo de entrega, as prioridades, e outros. O HTTP é o protocolo primário que deve ser usado como o protocolo de transporte, embora ele não seja um protocolo confiável. Atualmente vêm sendo empregados esforços para fornecer uma extensão confiável para o HTTP, como a especificação HTTP Reliable (HTTP-R), da IBM.

3.7.2.3 Contexto/privacidade

Os serviços web simples eram serviços de solicitação-resposta simples. Normalmente, em um ambiente comercial, apesar da mesma função comercial, é preciso alterar sutilmente o comportamento, para que as necessidades do usuário sejam atendidas. Por exemplo, um usuário pode ter preferências específicas, como o tipo de dispositivo, a linguagem, o país de origem, e outras. Portanto, é necessário incorporar um contexto na execução de um determinado serviço web.

Além disso, quando um consumidor de serviços chama um serviço web particular, é muito provável que o serviço web complexo seja, efetivamente, uma composição de vários outros serviços web. Esses serviços web podem ser serviços web adicionais dentro da empresa, ou também estar fora da empresa, o que confere uma nova dimensão à definição do contexto; isso significa que o contexto precisa ser passado (propagado), agora, a vários outros serviços web que funcionam harmoniosamente, de maneira que o processo de negócio seja completado.

Um contexto de serviço web geralmente está relacionado a informações do usuário e, portanto, é de extrema importância manter a privacidade dessas informações. Por exemplo, em um serviço web de validação de cartão de crédito, é preciso proteger as informações relativas aos cartões de crédito. O mesmo se aplica às transações financeiras que podem ocorrer nos serviços web. Portanto, o contexto caminha lado a lado com a privacidade.

Como uma observação secundária, a propagação das informações de contexto e a privacidade dessas informações não apresentam apenas desafios técnicos, mas também, legais. Por exemplo, para serviços web colaborativos, caso se tenha um servidor na Europa, a quantidade de informações que podem ser enviadas a partir desse servidor a um servidor localizado nos Estados Unidos é bastante restrita (regras da união européia).

Não existem, atualmente, padrões disponíveis para a propagação de informações de contexto, pois o contexto está fortemente relacionado a diferentes domínios industriais. As informações sobre o contexto, para um processo de negócio em uma indústria em particular podem não se ajustar adequadamente, umas às outras. Portanto, a abordagem recomendada consiste em fazê-lo dentro do seu próprio *site*. Entretanto, é importante enfatizar que é necessário um mecanismo extensível, para a propagação de contexto, de maneira que diferentes indústrias possam ser desenvolvidas com base nesse contexto.

3.7.2.4 Transações

As transações são fundamentais para qualquer negócio. Em poucas palavras, uma transação define uma unidade de trabalho que encapsula um conjunto de operações, de maneira que todas sejam bem sucedidas, ou que todas falhem. Se uma única operação critica não for bem-sucedida, então todas as operações relacionadas serão revertidas.

Um serviço web complexo provavelmente chama outros serviços web, conforme citado anteriormente. A natureza das transações em um serviço web complexo pode envolver transações de longa duração, e, portanto também se deve coordenar as transações através de serviços web que tenham sido chamados, o que contrasta com as transações curtas, encontradas nos bancos de dados e nos sistemas de transação típicos.

Atualmente, não existe um padrão disponível capaz de definir transações de serviços web no contexto das atividades e conversações entre serviços web colaborativos. Entretanto, um dos padrões emergentes é o Bussines Transaction Protocol (BTP).

3.8 Características de Acoplagem dos Serviços Web

A acoplagem indica a quantidade de folga, compensador ou passagem (*give*) entre dois itens. Caso os serviços sejam dependentes um do outro em um grau menor (ou frouxamente acoplados), eles podem se desenvolver independentemente um do outro.

Os benefícios de uma acoplagem mais folgada incluem:

- Uma maior reutilização.
- Uma maior produtividade.
- Torna-se mais viável a elaboração de sistemas a partir de simples partes.
- Diminuem as falhas em cascata.
- Diminuem os bugs, pois o que é reutilizado precisa de menos testes.
- Ficam mais fáceis os realces, modificações e reparos de bugs, pois as partes podem ser trabalhadas ou substituídas isoladamente.

A acoplagem mais folgada é um princípio muito importante nos serviços web, pois ela é muito flexível ao se trabalhar com diferenças tecnológicas, em qualquer dos lados de uma interação comercial.

A acoplagem mais folgada também permite a aplicação da lógica para se operar através das fronteiras organizacionais, onde as chamadas de procedimento remoto e os objetos distribuídos são insuficientes.

4 XML

A XML (eXtensible Markup Language - Linguagem de Marcação Estendida) é um subconjunto da SGML (Standard Generalized Markup Language - Linguagem de Marcação Padrão Generalizada) que permite que uma marcação específica seja criada para especificar idéias e compartilhá-las na rede. Ela tem as virtudes da SGML e da HTML sem qualquer das limitações óbvias. A XML permite intercâmbio de documentos, baseado em uma definição de metadados (*tags* auto definidos).

XML é uma recomendação da W3C (*World Wide Web Consortium* - comitê responsável pela formatação e evolução da linguagem XML) e o seu desenvolvimento e especificação tem sido supervisionado pelo XML Working Group. XML é uma especificação pública, ou seja, não é propriedade de nenhuma companhia. O desenvolvimento de XML começou em 1996 e é um padrão W3C desde fevereiro de 1998.

Em traços muito gerais o HTML é identificado por um conjunto de códigos que permitem representar graficamente um determinado documento, através de uma formatação baseada num esquema de etiquetas (*tags*) que, posteriormente, são interpretadas pelos *browsers*. Tal como o HTML, o XML também utiliza etiquetas como <*title*> ou <*body*> . Mas, enquanto que com o HTML é a própria linguagem que especifica o que cada etiqueta e atributos especificam, o XML usa as etiquetas apenas para delimitar blocos de dados e deixa a interpretação dos mesmos para a aplicação. Assim, em função do contexto programático, um <p> que em HTML significa sempre um novo parágrafo, pode em XML significar um preço, uma pessoa, ou qualquer outra coisa conforme definido.

Apesar de ser uma ferramenta fácil de usar e aprender, fato pelo qual se generalizou bastante o seu uso, verificam-se algumas desvantagens na utilização do HTML que se prendem ao fato dos dados estarem dependentes da sua apresentação gráfica; não fornecer conteúdo semântico e ter um número limitado de etiquetas.

O HTML especifica a formatação gráfica de um documento, enquanto que o XML especifica a estrutura de dados. O XML não se prende a etiquetas predefinidas, sendo possível criar qualquer estrutura e definir uma interpretação para essa estrutura.

O XML é uma tecnologia de conteúdos alternativa e também complementar ao HTML, que vai provavelmente rivalizar mais com os formatos proprietários de armazenamento de documentos, do que propriamente com o HTML. Por ser altamente adaptável aos dados que pretende descrever, por facilitar a interação entre aplicações e documentos, e por produzir documentos autodescritivos, a tecnologia XML tem condições para se tornar num *standard* para a publicação, armazenamento e transferência de documentos eletrônicos, o que, aliás, já vem se concretizando.

Relativamente ao HTML, o XML trás grandes vantagens na sua utilização. Para além da possibilidade de definição de novas marcas e etiquetas à medida dos utilizadores e de um maior controle sobre a aplicação de formatações e da forma como estas aparecem nos *browsers*, o utilizador do XML fica com a certeza de publicar os seus documentos num formato *standard* (independente dos fabricantes de software), facilmente interpretável por múltiplas aplicações e, em certa medida, auto-explicativo. W3C (16) XML (15)

4.1 Histórico

Desde a invenção da impressão os escritores tomam notas em manuscritos para instruir os impressores a respeito da formatação de caracteres e outras questões de produção. Estas notas se chamavam marcações (*markups*), e uma coleção destas anotações observam uma sintaxe e gramática definidas e podem ser chamadas de linguagem. Por exemplo, revisores usam uma linguagem de marcação (*markup language*, ML) para informar os autores quanto a correções. Mesmo o uso moderno de pontuações é uma forma de marcação que permanece com o texto para avisar ao leitor sobre como interpretar o texto.

Em 1969 uma pessoa andou na lua pela primeira vez. Naquele mesmo ano, Ed Mosher, Ray Lorie e Charles F. Goldfarb, da IBM Research, inventaram a primeira linguagem de marcação moderna, a General Markup Language (GML). A GML era uma linguagem de referencial própria para a formulação de estruturas de um conjunto arbitrário de dados, e seu propósito era ser uma meta-linguagem - uma linguagem que pudesse ser usada para descrever outras línguas, suas gramáticas e seus vocabulários.

A GML se tornou mais tarde Standard Generalized Markup Language (SGML). Em 1986, a SGML foi adotada como um padrão de troca e armazenagem de dados internacional pela *(International Organization for Standardization* -ISO), chamada IS0-8879.

Markup é um método de se transportar metadados (isto é, informações sobre um conjunto de dados). Linguagens de marcação usam *string* literais, ou "*tags*", para delimitar e descrever estes dados.

Em 1996, o World Wide web Consortium (W3C) começou o projeto de uma abrangente linguagem de marcação que combinaria a flexibilidade e capacidade do SGML com a ampla aceitação do HTML.

A linguagem que se transformou no XML teve como base às especificações do SGML, e realmente, foi especificada para ser um sub-conjunto desta linguagem. O SGML já fornecia uma linguagem aberta que poderia ser expandida por qualquer um com qualquer finalidade.

A idéia de que o XML deveria ser mais simples que o SGML foi motivada por considerações sobre facilidade de uso: em parte a leitura e digitação de linguagem de marcação por usuários de ferramentas amplamente disponíveis e simples, mas também a simplificação do processo de computação de documentos e intercâmbio de conjuntos de dados.

Devido às suas muitas propriedades opcionais, o SGML é tão complexo que é difícil escrever *parsers* genéricos, enquanto *parsers* em XML são muito mais simples. Além disso, o XML alavanca protocolos e softwares da Internet já existentes, visando à facilidade de transmissão e processamento de dados.

A XML 1.0 se transformou em uma recomendação do W3C em fevereiro de 1998. A especificação oficial, incluindo a gramática em notação *Extended Backus-Naur Form* (EBNF), está amplamente disponível na Internet no site da W3C.

A XML é uma maneira simples e padrão de delimitar os dados do texto. Foi descrito como o "ASCII da web". É como se fosse possível a cada usuário usar a sua linguagem de programação favorita para criar uma estrutura arbitrária de dados, e então compartilhá-la com qualquer pessoa usando outra linguagem em qualquer outra plataforma de computação.

As *tags* do XML dão nome ao conceito que se está descrevendo, e atributos identificados modificam as estruturas rotuladas. Assim, pode-se descrever oficialmente a sintaxe desenvolvida e compartilhá-la com outros.

Eis um exemplo simples:

```
<?xmlversion="1.0"?>
<livros>
<livro categoria="Tecnica">
<autor>Maidi Terezinha Dalri</autor>
<titulo>serviço web no mundo Java</titulo>
<preco>81.95</preco>
</livro>
<livro categoria="Educacao">
<autor>Maidi Terezinha Dalri</autor>
<titulo>A participação da informática no processo educativo </titulo>
<preco>12.99</preco>
</livro>
</livros>
```

Este mecanismo de descrição de dados é XML, e significa que este é um ótimo modo de compartilhar informação via Internet porque:

- É aberto; o XML pode ser usado para trocar dados com outros usuários e programas em uma plataforma de maneira independente.
- Sua natureza auto descritiva o torna uma opção eficaz para soluções de empresa-a-empresa e extranet.
- É possível compartilhar dados entre programas sem coordenação prévia. Os mecanismos em XML permitem que se descubra a estrutura de uma classe de documentos XML.

Para trabalhar com documentos XML, o W3C padronizou uma API (Aplicação de Interface de Programação) para o XML, assim é fácil criar programas que lêem e escrevem XML.

Além disso, a XML foi projetada para oferecer suporte imediato para linguagens não-européias e internacionalização. Assim como HTML 4.01, a XML se baseia no *Universal Character Set -UCS* (conjunto de caracteres universal), definidos no padrão de conjunto de caracteres ISO/IEC 10646 (que atualmente é compatível com o padrão um pouco mais conhecido *Unicode* (www.unicode.org)). Todas as características que levaram o HTML à popularidade estão presentes no XML. W3C (16) XML (15)

4.2 Características da XML

4.2.1 Extensível

Ao contrário da linguagem HTML, que dispõe de um determinado conjunto de TAGS definidos, na XML pode-se definir TAGS próprios específicos do problema sendo modelado, o que abre ilimitadas possibilidades.

4.2.2 Estruturada

XML define a estrutura de um dado através de seus *tags*. A criação de novos *tags* deve seguir as regras definidas no padrão, que são bastante simples, mas suficientes para garantir a estruturação adequada da meta-informação.

O nome do autor é facilmente identificável no exemplo citado anteriormente:

<autor>Maidi Terezinha Dalri </autor>

Os tag XML devem seguir o conceito de "*well-formed tags*", ou seja, *tags* bem formados, que atendam ao formato definido para *tags*. As especificações completas dos critérios de um XML bem formados encontram-se especificados no site do W3C, dentre elas:

- Todo *TAG* deve ter o seu correspondente que o encerra.
- Se o *TAG* contém atributos, então os valores dos atributos devem estar contidos em aspas ("xxx").
- É permitido e importante o aninhamento de *TAGS* e deve seguir a regra: "último *TAG* aberto é o primeiro *TAG* a fechar". A endentação serve apenas para facilitar a leitura humana, sendo ignorada pelo programa leitor do XML.

```
<cliente especial="sim">
<nome>Jose da Silva</nome>
      <idade>45</idade>
      <sexo>Masculino</sexo>
      <endereco>
            <tipo>Trabalho</tipo>
            <rua>Rua ABC</rua>
            <numero>150</numero>
            <bairro>Atalaia</bairro>
            <apto/>
      </endereco>
      <endereco>
            <tipo>Trabalho</tipo>
```

```
            <rua>Rua XYZ</rua>
            <numero>250</numero>
            <bairro>Atalaia</bairro>
            <apto/>
        </endereco >
    </cliente>
```

- Se um *TAG* está vazio ele pode utilizar:
  - a versão simplificada do *TAG* <apto/> ou
  - a versão completa do *TAG* <apto></apto>

4.2.3 Validação

Utilizando um arquivo DTD – *Document Type Definition*, pode-se especificar regras para a criação de um determinado XML, essas regras seguem a regra de negócio que estruturam os dados.

Por exemplo: Pode-se definir que todo cliente é obrigado a ter um nome e que a idade é opcional. Assim, ao se criar um XML que não siga essa definição o programa poderá detectá-lo facilmente.

4.3 A Estrutura da XML

As *tags* que delimitam o conteúdo do XML identificam o elemento específico dos dados que delimitam. Atributos identificados são encontrados nas *tags* que oferecem dados adicionais sobre o elemento descrito. Por exemplo:

```
<contato codigo="33">
<nome>Maidi Terezinha Dalri</nome>
<email>maidi@setrem.com.br</email>
<telefone tipo="celular">
<area>55</area>
<numero>9977 0196</numero>
</telefone>
</contato>
```

Os dados são auto descritivos à medida que cada item leva seu próprio nome que pode estar relacionado a um modelo externo para problemas do mundo real que o documento descreve. Até aqui o código se parece em muito com HTML. O significados das *tags* HTML, contudo, é estipulado pelo W3C.

O XML oferece ao desenvolvedor a possibilidade de definir as suas próprias *tags* à medida e conforme as suas necessidades. No entanto, há um conjunto de regras que têm de ser seguidas a fim de se obter como resultado um documento bem formado, ou seja, um documento que respeita integralmente as regras gramaticais definidas. Esta restrição permite o processamento do XML de forma muito mais eficiente do que no HTML, a simplificação do software de processamento e a representação de uma estrutura de dados de forma inequívoca. W3C (16) XML (15)

4.4 Vocabulários XML

Como já citado anteriormente neste trabalho, uma das propriedades mais significativas do XML é sua inerente extensibilidade. Em comparação, a HTML começou como uma simples linguagem de marcação (com um conjunto de *tags* definido) para papéis científicos de forma a permitir sua troca pela Internet, mas se desenvolveu rapidamente à medida que desenvolvedores de browser adicionaram novas propriedades e *tags* a seus softwares. Muitas destas adições à HTML eram para dar suporte a propriedades multimídia e páginas comerciais vistosas da web. Infelizmente, essas *tags* eram criações semiparticulares de empresas individuais, e freqüentemente causavam problemas a outros browsers. Algumas adições se tornaram parte oficial do HTML, mas muitas ainda são propriedade particular.

A XML, por outro lado, sempre visou permitir a construção fácil e rápida de conjuntos de tag específicos para empresas, disciplinas científicas e outros domínios. Enquanto cada empresa (ou cada indivíduo) pode optar por definir seus próprios conjuntos de tag XML, uma das forças do XML é o compartilhamento de tais "vocabulários", todos eles usando a mesma sintaxe básica, *parsers*, e outras ferramentas.

Vocabulários XML compartilhados oferecem documentos e bases de dados mais fáceis de serem pesquisados, e uma maneira de se trocar informações entre muitas organizações e aplicações de computadores diferentes.

Um vocabulário eficiente descreve os objetos relevantes no seu problema de maneira a exprimir o problema permitindo ao mesmo tempo, o acesso fácil aos seus dados.

Um vocabulário XML é uma descrição de dados XML usados como meio de troca de informação, freqüentemente dentro de um domínio específico de atividade humana (negócios, química, direito, música, por exemplo).

4.4.1 Vocabulários científicos

A *Chemical Markup Language* (CML) é um vocabulário voltado para a área de química, tem sido mencionada como o "HTML com moléculas", mas o CML também oferece a conversão de vários formatos de arquivos proprietários sem perda de semântica, e a criação de documentos estruturados apropriados para publicações profissionais.

A linguagem fundamental da ciência é a matemática, e há um vocabulário XML chamado MathML que oferece uma maneira de se trocar expressões matemáticas. MathML substituirá a exibição de equações como meros retratos e/ou feiosas aproximações ASCII, permitindo uma renderização sofisticada com browsers apropriados, e oferecerá um formato de troca para álgebra, geometria, estatística simbólica e outras ferramentas de software de matemática.

Outros vocabulários científicos incluem a *Bioinformatic Sequence Markup Language* (BSML), para a quantidade maciça de informações produzidas pelo mapeamento e sequenciamento genético e a *Instrument Markup Language* (IML) da NASA para o controle de instrumentos de laboratório e sua primeira implementação, *Astronomical Instrument Markup Language*. Essas IMLs são usos clássicos dos vocabulários XML para documentos técnicos estruturados e a disseminação precisa de informações técnicas e científicas. Tal uso do XML é muito promissor também para poderosas ferramentas de educação. Informações e documentação sobre esses e os demais vocabulários aqui apresentados podem ser encontrados no site da W3C.

4.4.2 Vocabulários de negócios

A aplicação mais popular da computação é o comércio. O sangue financeiro do mundo flui pelas veias das redes de computador usando uma variedade de formatos de dados e esses formatos de dados são, na sua maioria, particulares.

O comércio troca tanto produtos quanto dinheiro, e tais trocas são descritas como transações. Estas transações envolvem freqüentemente a troca de documentos legalmente passíveis de vinculação. É com freqüência que esses documentos são trocados eletronicamente, usando padrões *Electronic Data Interchange* (EDI). O EDI define um formato para muitas transações entre empresas que são a base da maior parte do comércio. A origem do EDI na América do Norte pode ser traçada até o Comitê de Coordenação de Dados de Transporte no início dos anos 70. No início dos anos 90, a ANSI lançou os padrões X12 (conhecidos como "ASC X12"). O desenvolvimento destes padrões nos EUA é supervisionado pela *Data Interchange Standards Association* (DISA).

O *Open Buying on the Internet Consortium* é uma organização sem fins lucrativos que está desenvolvendo padrões abertos para trocas entre empresas pela Internet. Apesar de o OBI v2.0 se basear nos padrões ASC X12, ele está sendo retrabalhado como um vocabulário XML. Enquanto isso, a Ariba e a Microsoft desenvolveram o *CommerceXML* (cXML) com propósitos similares e a *CommerceOne* oferece a *Common Business Library (*CBL).

Outra iniciativa industrial, com amplo apoio, é a organização RosettaNet (www.rosettanet.org), que descreve a troca de informações entre empresas usando o XML, UML e outros protocolos comuns. A base deste projeto são dois dicionários principais: primeiro, um conjunto de especificações técnicas para todas as categorias de produtos; e depois, descrições de empresas e transações comerciais.

Alguns anos atrás, a Microsoft, Intuit e CheckFree se uniram para desenvolver uma especificação aberta para transferências *online* de dados financeiros, chamada *Open Financial Exchange* (OFE, ou OFX), baseada no SGML. Porém, o OFX nunca foi SGML totalmente válido, uma vez que permitia a inclusão de elementos não definidos. Em 1998, a migração para o XML foi considerada uma meta para o OFX. Na primavera de 1999, o OFX e seus relativos padrões migraram para um novo ML, chamado *Interactive Financial Exchange* (IFX).

4.4.3 Vocabulários jurídicos

A representação digital de formulários em papel permanece um problema para comunidades comerciais, jurídicas e médicas. Uma possível solução é o *Extensible Forms Description Language* (XFDL). Este vocabulário XML oferece suporte à precisão de *layout*, computações, validação de entradas, assinaturas digitais, e registros de transações legalmente vinculantes e rastros para auditoria.

4.4.4 Vocabulários médicos

As informações médicas cobrem as gamas de usuários de XML. Materiais de referência médica e papéis relacionados à ciência que usam marcação XML podem ser facilmente traduzidos para apresentações em vários formatos; e são mais prontamente encontrados e recuperados usando buscas estruturadas mais focadas e poderosas do que buscas livres de texto do tipo booleanas. Informações clínicas, financeiras e administrativas precisam ser trocadas entre vários sistemas independentes de computadores, em várias organizações diferentes, indo desde hospitais e farmácias até seguradoras e/ou agências do governo. Em 1987, uma iniciativa ANSIX12 produziu o padrão *Health Levei 7* (HL7) que é usado na maioria dos grandes hospitais americanos hoje em dia, bem como na Austrália,

Europa Ocidental, Israel e Japão. Apesar deste padrão não ser atualmente implementado usando XML, tem sido alcançado progresso em direção a uma versão XML.

4.4.5 Vocabulários de computação

A Internet e a WWW também precisam de meios para descrever a troca de informações de fontes as mais diversas e de vários formatos.

Um dos primeiros vocabulários XML é o *Channel Definition Format* (CDF), um formato proposto pela Microsoft para permitir que web sites ofereçam coleções de informações atualizadas com frequência (chamadas "canais") para a entrega automática a assinantes. Infelizmente, a submissão do CDF ao W3C tem mais de 2 anos e meio, e não é mais compatível com XML (já que se baseava num rascunho antigo das especificações do XML e inclui agora sintaxe ilegal).

Há o *Structured Graf Forma* (SGF), um formato XML para descrever a estrutura de sites da Internet baseado em formatos matemáticos de estrutura de grafos.

O projeto Maxila da Netscape utilizou-se de uma *XML-based User Interface Language* (XUL) como uma maneira independente de plataforma para descrever interfaces de usuário. O XUL pode conter tipos de elementos para controles UI, marcação HTML4 para dados e JavaScript para eventos de usuários.

O *Bean Markup Language* (BML) da IBM é uma configuração ML com base XML de componente personalizado para o modelo de componente JavaBean. O BML pode ser usado para descrever a criação de novos JavaBeans, acessar e/ou configurar JavaBeans existentes, vincular eventos de um JavaBean a outro, e chamar métodos arbitrários em outros JavaBeans .

A distribuição de *software* em uma rede poderia ser controlada usando o formato *Open Software Description* (OSD) proposto pela Marimba e Microsoft.

O design moderno de bases de dados usa um processo rigoroso baseado na modelagem de dados, usando freqüentemente a *Unified Modeling Language* (UML). O *Meta Object Facility* (MOF) é um padrão da *Object Management Group* (OMG) para a administração de repositórios e metadados distribuídos. A OMG, grandes empresas de sistemas (como a IBM e Unisys) e empresas de software de bases de dados (como a Oracle, Rational e Sybase) têm promovido a fusão do XML, UML e MOF na forma da especificação *XML Metadata Interchange* (XMI). Ainda que este não seja um vocabulário XML, estritamente falando (uma vez que se trata de um superconjunto de XML), o XMI é um bom exemplo do poder do XML para outras propostas que não sejam documentos.

Um dos domínios tecnológicos mais conservadores é a rede de telefones públicos Por anos, esta rede tem usado um protocolo bastante complexo, o *Signaling System 7* (SS7). Recentemente, várias alternativas com base no XML foram propostas, incluindo a *Call Policy Markup Language* (CPML). Este é um subproduto de uma tendência em direção de padrões mais abertos na indústria da telecomunicação privada.

Muitos destes vocabulários foram ou ainda estão sendo definidos em colaboração por grupos de empresas unicamente com o propósito de poder trocar dados prontamente. XML (15)

4.5 XML Bem-Formado X XML Válido

Basicamente, o XML bem-formado obedece aos requisitos de sintaxe da Recomendação XML 1.0 do W3C, ou seja devem obedecer as regras de construção de documentos XML genérico, quais sejam:

- Ter um e apenas um elemento raiz.
- Valores dos atributos estarem enter aspas ou apóstrofes.
- Atributos não se repetirem.
- Todos os elementos terem etiqueta de fechamento.
- Elementos estarem corretamente aninhados.

A validação de um documento dá-se em relação à definição de um DTD ou de um esquema. Um documento XML é considerado válido em relação a um esquema se obedecer todas as suas regras.

Os *parsers* são ferramentas de processamento que verificam se um documento obedece às regras de sintaxe do XML.

Um *parsers* de não validação somente verifica o documento quanto à boa formatação em relação às regras centrais da sintaxe XML. Um *parser* de validação verifica também um documento contra um DTD para decidir se este é legítimo de acordo com as regras no DTD.

4.6 Definições de Tipos de Documentos

Document Type Definition (DTD) constitui-se num conjunto de definições que capturam as regras que um designer adiciona para ampliar as regras centrais da sintaxe XML e criar um vocabulário para descrever um problema ou situação. Trata-se do primeiro mecanismo para a descoberta de um vocabulário de estrutura de XML.

Os DTDs seguem suas próprias regras sintáticas, mas permitem a elaboração de sentenças bem definidas sobre o que é ou não um documento permissível em uma determinada classe de documentos XML. Isto leva diretamente a uma distinção entre *parsers* de validação e não validação.

Para se definir um arquivo DTD devem ser realizadas duas coisas:

- Criar o arquivo DTD
- Associar o DTD a arquivo XML

Para criar o arquivo DTD é importante conhecer a estrutura do XML.

4.7 Namespace e Schemas

Um *namespace* é uma fonte de nomes nos quais um desenvolvedor de um documento quer trabalhar. Ao atribuir o recurso, pode-se utilizar de outros recursos e usar vocabulários parciais sem ambigüidade.

Os DTDs não permitem o uso de *namespaces*. Os DTDs têm outros problemas e, para ambos a solução tem sido os *schemas*, uma substituição para DTDs usando a sintaxe do XML.

4.8 API's XML

Um módulo de software capaz de ler documentos e fornecer acesso a seu conteúdo e estrutura é chamado de XML parser e a interface de programação, incluindo os nomes dos métodos e atributos é uma API XML. Desenvolvedores são livres para implementar suas próprias APIs, levando em consideração os padrões aceitos pela indústria. Fazendo isto, um desenvolvedor pode escrever uma API que pode ser executada em diferentes ambientes sem maiores modificações.

Existem varias interfaces de programação (APIs) desenvolvidas ou em desenvolvimento para a manipulação de XML, porém, as duas principais especificações são: SAX (*Simple API for XML*) e DOM (*Document Object Model*).

4.8.1 Document object model

O *Document Object Model* (DOM) é um API que especifica um conjunto de objetos e interfaces para manipular documentos HTML e XML. O W3C mantém a Recomendação do Document Object Model (DOM), e é um dos dois APIs que mais têm suporte, para trabalhar com documentos XML.

O DOM oferece uma visão do documento estruturada em árvore. Um *parser* compatível com DOM lê todo o documento e oferece uma visão construindo uma árvore de

objetos na memória. As principais estruturas do componente do documento são nós na árvore objeto. Acessar itens e manipulá-los é uma questão de navegar a árvore de análise usando as interfaces DOM.

O DOM é orientado a objetos criados a partir do XML lido.

Classes Java podem utilizar a API de manipulação do XML no modelo DOM para obter, alterar e navegar pelas informações do arquivo.

4.8.2 Simple API for XML

Simple API for XML (SAX), ao contrário do DOM, o SAX não é o produto de uma organização de padrões. É o produto casual de um número de desenvolvedores de XML que precisavam de um API eficaz no início do desenvolvimento do XML. O SAX permanece popular porque tem uma abordagem diferente para oferecer o acesso a documentos XML. Ao invés de apresentar aplicações com uma visão em árvore do documento completo, o SAX oferece eventos à medida que analisa o documento. Os eventos tem a forma de uma *tag* inicial. Um *parser* compatível com SAX não faz esforço para reter o documento; ao invés disso, avisa ao programa usado à medida que processa cada parte do documento. 0 que acontece em resposta a um evento fica por conta do programa usando o *parser*. Um programa tem total responsabilidade pela manutenção do estado do documento. Pode manter tanta ou tão pouca informação quanto seja preciso para satisfazer os requisitos que motivam a aplicação.

Isto origina um parser bastante compacto que faz pedidos mínimos aos recursos do sistema, sendo portanto ideal para lidar com documentos XML muito grandes.

O SAX é orientado a eventos que ocorrem durante a leitura seqüencial do arquivo XML.

Classes Java que implementam a API para manipulação de XML no modelo SAX são notificadas a cada evento ocorrido e tomam as ações necessárias.

Modelo adequado para a manipulação de arquivos XML grandes.

4.9 Links e Consultas em XML

Uma das características do HTML é a ligação (*linking*). É provavelmente a característica individual mais popular desta linguagem de marcação. Bases de dados relacionais formam, elas mesmas, uma ligação, usando chaves diferentes para trazer dados de outra tabela. Qualquer tecnologia que espera fazer parte de aplicações consistentes precisa ter provisões para ligar corpos de dados. O XML não está imune aos charmes da ligação. Muitos desenvolvedores querem ligar um documento XML a outro, ou ligar conteúdo

não XML a um documento XML. Imagens e outros dados binários poderiam acompanhar um documento XML caso se tenha *links* no XML. A ligação é uma das áreas principais de investigação para a comunidade XML. Várias propostas têm sido trabalhadas através do W3C, a saber, o XLink e XPointer.

4.10 Transformação do XML

Uma das poderosas técnicas do XML é a transformação. A transformação permite que um programador mapeie um documento XML de um formato para outro, baseado em um conjunto de regras aplicadas ao primeiro documento. As transformações do XML são usadas para a tradução entre vocabulários XML similares, assim como traduzir documentos XML para outros formatos de arquivo com base em texto, assim como valores separados por vírgulas. Isto é uma ferramenta importante para os desenvolvedores da web. No caso de se utilizar de forma conjunta recursos existentes, pode ser necessário executar algumas transformações eficientes para obter um formato comum. O interessante sobre transformações é que o mapeamento é especificado em um documento separado, ao invés de um código. Se precisar traduzir dinamicamente entre uma série de formatos relacionados, pode desenvolver uma série de documentos de regras. Em tempo de funcionamento, pode determinar que transformação é necessária e simplesmente aplicar o conjunto de regras apropriadas ao documento em uso. Isto é especialmente útil em aplicações de negócio para negócio e cadeias de suprimento.

O formato mais recente das transformações é parte de uma linguagem de estilo para XML chamada *Extensible Style Language* (XSL). Na verdade, a linguagem de transformação é chamada *XSL Transformation* (XSLT). O propósito básico da XSLT é organizar um documento XML para ser usado com o estilo XSL. Ainda que não tenha sido projetada como uma linguagem de transformação com propósitos genéricos, a XSLT é bastante flexível e permitirá que se execute a maioria das traduções, classificando e organizando tarefas em XML sem escrever seu próprio código procedural. Ao invés disso, são escritas regras para transformar XML com base no contexto no qual os elementos aparecem.

O estilo em XML é uma ferramenta importante para desenvolvedores da web.

As técnicas usadas para o estilo em XML variam em sua complexidade e sofisticação. Talvez as técnicas mais simples sejam as *Cascading Style Sheets* (CSS). São básicas e ao mesmo tempo capazes de atribuir determinadas informações sobre estilo a elementos XML identificados. Isto ocorre de maneira bastante parecida como a forma com que um processador de palavras atribui detalhes sobre fonte, distância, cor e outros a estilos identificados no documento.

4.11 Aplicabilidade

A XML permite oferecer aos programadores ferramentas para construir sistemas distribuídos em camadas múltiplas, mantidas juntas através de dados de padrões abertos, autodescritivos. A XML não trata de comunicações servidor a servidor em si, mas as partes inovadoras dentro da comunidade XML usaram esta tecnologia para formalizar tais comunicações. Uma vez que os dados que um servidor receberia em tal troca também é XML, é bastante fácil para o primeiro servidor fundir vários documentos ou transformar um documento em outro formato para satisfazer um pedido. O mesmo mecanismo que o cliente pode usar para descobrir estrutura está disponível nos servidores. Uma vez que XML é inerentemente hierárquico, ele pode facilmente decodificar fontes de dados não relacionais.

A XML é composta por dados e prontamente manipulado por programas. Os mesmos dados podem receber estilo para apresentação no *browser* ou podem estar sujeitos ao pós-processamento através de um agente. O mecanismo de troca -documentos XML não presume nada quanto ao uso final dos dados. Se o cliente é conhecido por pedir HTML, uma transformação movida pelos dados pode ser aplicada a um documento XML para criar uma página HTML.

O casamento entre a aplicação do servidor e o cliente é consideravelmente solto devido a habilidade de um programa de descobrir a estrutura de um documento XML. Assim, aplicações ambiciosas podem ser escritas para criar documentos com estruturas originais dependendo do estado da aplicação, que ainda pode ser interpretado sem se ter que escrever software para cada novo tipo de estrutura de documento.

Os dados em XML podem:

- ser distribuídos em outras aplicações, objetos ou servidores *middleware* para serem posteriormente agregados, processados e trocados;
- ser editados e manipulados localmente pelas aplicações do cliente. XML pode localizar e processar dados que permitam, por exemplo, contabilizar e hierarquizar as obras de um autor, por ano de edição, sem chamadas adicionais ao servidor;
- visualizados de formas diferentes, após distribuídos no *desktop*. Os dados podem ser apresentados de forma dinâmica, de acordo com as necessidades do cliente. Um investigador sobre Camões pode querer visualizar os campos autor, título, ano, editora e resumo de uma obra, ao passo que um arquivista que pretenda importar o registro da mesma obra para a sua Base de Dados de Gestão Biblioteconômica, vai querer visualizar não só alguns mas todos os campos de um registro de acordo com uma hierarquia pré-definida por si;

- atualizados e compostos de forma desagregada. Se novos registros sobre o autor Luís de Camões forem introduzidos, podem ser estes, e só estes dados, a serem enviados do servidor ao cliente. Ao contrário do que acontece atualmente, no caso da contabilização das obras sobre Luís de Camões, por ano de edição, apenas o pedaço de informação respectivo seria reconstruído – e não toda a página. São também possíveis de acrescentar resultados de cálculos probabilísticos, como a evolução da média anual dos registros acerca da obra deste autor.

Sendo uma meta-linguagem, XML também torna possível a criação de novas linguagens que padronizam o tratamento de qualquer tipo de informação, como por exemplo a apresentação de dados (XSL), a transferência de dados (SOAP), a comunicação entre aplicações (WSDL), a descrição de relacionamento entre dados (Xpointer, Xlink), a descrição de aplicações (AppML) entre outros.

Para a implementação de serviços web, XML fornece um meio de troca de informações a respeito de interfaces, tipos de parâmetros e resultados, devido à sua natureza independente de linguagem ou plataforma específica, podendo ser considerada a torre de babel tecnológica que deu certo.

5 SOAP

O *Simple Object Access Protocol* (SOAP) é um protocolo de computação distribuído que permite a troca de informações em um ambiente descentralizado e distribuído. Entretanto, o SOAP não define a semântica das mensagens, fornece, apenas, o framework. Outras especificações, como a eb-XML, que utiliza o SOAP como seu mecanismo de transporte, definem a semântica das mensagens. Em muitos casos, essa semântica é definida na aplicação.

O SOAP se comunica com sistemas distribuídos usando o protocolo XML baseado em texto, em vez de um formato binário usado por outros protocolos de computação distribuídos, como o CORBA, o RMI e o DCOM. Essa característica torna o SOAP altamente interoperável por meio de várias plataformas de *hardware*, sistemas operacionais, linguagens de programação. O SOAP pode ser transportado pelo HTTP, o que permite que se beneficie dos investimentos em infra-estrutura, como servidores web, servidores *proxy* e *firewalls*. O SOAP também pode ser transportado usando outros protocolos, como SMTP e JMS.

A versão do SOAP sobre a qual esse trabalho versa é a 1.1, mas no momento da conclusão deste trabalho o protocolo já se encontra na versão 1.2.

O protocolo SOAP é considerado superficial, pois contém menos recursos do que outros protocolos de computação distribuídos, o que torna o protocolo menos complexo. Por exemplo, diferentemente do CORBA, o SOAP não depende de um *Object Request Broker* (ORB). Isso significa que os sistemas que permitem o CORBA devem se comunicar através do ORB. Entretanto, o SOAP permite que sistemas distribuídos se comuniquem diretamente, sem nenhum intermediário. Além disso, o SOAP não define um mecanismo para a descrição e localização de objetos em sistemas distribuídos. Esses mecanismos são definidos pelas especificações WSDL e UDDI apresentados nesse trabalho nos capítulos 6 e 7.

A especificação SOAP consiste em duas partes: encapsulamento de mensagens e de RPC (*Remote Procedure Call*, ou chamada de procedimento remoto).

A parte da especificação relativa às mensagens define como embutir chamadas e respostas de procedimento remoto dentro das mensagens, para efeito do chamamento de procedimentos em sistemas remotos.

A parte da especificação relativa ao encapsulamento da RPC permite que o SOAP forneça uma abstração na camada do protocolo, para protocolos de computação distribuídos dentro de uma intranet ou na Internet. Essa abstração fornece sistemas remotos com acesso a objetos através de plataformas, sistemas operacionais e ambientes que seriam, de outra maneira, incompatíveis. Por exemplo, uma camada de SOAP pode ser adicionada a um objeto java e CORBA. Essa camada cria um ambiente de comutação distribuído comum para os dois objetos tornando os objetos do CORBA acessíveis pelos objetos java e vice-versa, sem que seja preciso se preocupar com as peculiaridades de cada protocolo de computação distribuído. A figura 5.1 representa isso:

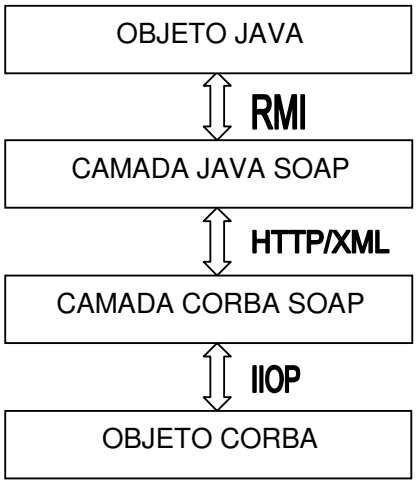

Figura 5.1. Comunicação através do SOAP

Os pedidos SOAP podem ser feitos em três padrões: GET, POST e SOAP. Os padrões GET e POST são idênticos aos pedidos feitos por navegadores Internet. O SOAP é um padrão semelhante ao POST, mas os pedidos são feitos em XML e permitem recursos mais sofisticados como passar estruturas e *arrays*. Independente de como seja feito o pedido, as respostas são sempre em XML. O XML descreve perfeitamente os dados em tempo de execução e evita problemas causados por inadvertidas mudanças nas funções, já que os objetos chamados têm a possibilidade de sempre validar os argumentos das funções, tornando o protocolo muito robusto.

O SOAP pode ser facilmente implementado em virtualmente qualquer ambiente de programação. Existem atualmente diversos "*kits*" de desenvolvimento SOAP para vários sistemas operacionais e linguagens de alto nível.

O SOAP está estruturado em três partes a saber:

- A construção de envelope SOAP define uma estrutura global para expressar o que está na mensagem; quem deveria negociar com ela, e se ela é opcional ou obrigatório.

- As regras de codificação SOAP definem um mecanismo de codificação do documento que pode ser usado para instâncias de troca de tipos de dados de definição da aplicação.

- A representação RPC SOAP define uma convenção que pode ser usada para representar chamadas de procedimento remoto e respostas.

A definição mais clara para o fácil entendimento do SOAP é de que o mesmo possibilita a utilização de métodos de um objeto através da web. O SOAP nada mais é do que um protocolo que possibilita a execução de métodos com passagem e retorno de parâmetros através de um arquivo XML que trafega pela web via HTTP. W3C (16)

5.1 A História do SOAP

Um grupo de pessoas em empresas como a Microsoft, a DevelopMentor e a UserLand Software esteve envolvido na criação do SOAP, em 1998. A intenção original do protocolo consistia, simplesmente, em definir um mecanismo para a transmissão de documentos XML que contivessem comandos capazes de disparar operações ou respostas em sistemas remotos, ou RPC.

Para atingir esse objetivo seriam necessários uma linguagem de esquema ou um sistema de tipos para XML.

Até então, isso não existia. Portanto, suas atenções se voltaram para a definição de um sistema de tipos. Mas o sistema de tipos era elementar; contendo alguns tipos, estruturas e arrays primitivos. De acordo com um artigo escrito por Don Box, o SOAP teria sido lançado como uma especificação em 1998, mas interesses políticos de grupos da Microsoft que estavam trabalhando na especificação condenaram a especificação ao ostracismo, até que as divergências políticas fossem resolvidas.

Cansado de esperar pelo desenrolar do processo, Dave Winer, presidente da UserLand Software e uma das pessoas que estavam trabalhando na especificação SOAP, decidiu produzir uma especificação similar, denominada XML-RPC. Essa especificação utiliza um subconjunto do sistema de tipos que foi definido como o sistema de tipos original para o SOAP.

No quarto trimestre de 1999, a especificação SOAP 1.0 foi remetida. Mas o sistema de tipos nativo foi substituído pela especificação "XML Schema Part 2 Datatypes", que definiu tipos de dados do XML. A maioria da especificação SOAP 1.0 indicava como modelar dados usando o novo sistema de tipos. Não se registrou alterações consideráveis na especificação, entre o SOAP 1.O e o SOAP 1.1.

A especificação SOAP 1.1 foi a primeira versão a ser submetida ao *World Wide web Consortium* (W3C). Foi usada para efeito de discussão e para decidir se um grupo de trabalho W3C poderia ser estabelecido para modelar o protocolo como um padrão formal. Decidiu-se criar um grupo de trabalho para o protocolo, sendo o primeiro esboço de trabalho da especificação o SOAP 1.2, Working Draft 1. O *release* atual do SOAP é a versão 1.2. W3C (16)

5.2 Vantagens do SOAP

O protocolo SOAP tem diversas vantagens sobre outras maneiras de chamar funções remotamente como DCOM, CORBA ou diretamente no TCP/IP:

- É simples de implementar, testar e usar.
- É um padrão da indústria, criado por um consórcio, adotado pela W3C http://www.w3.org/TR/SOAP/) e por várias outras empresas.
- Usa os mesmos padrões da web para quase tudo: a comunicação é feita via HTTP com pacotes virtualmente idênticos; os protocolos de autenticação e encriptação são os mesmos; a manutenção de estado é feita da mesma forma; é normalmente implementado pelo próprio servidor web. O fluxo dessa comunicação pode ser visualizado na figura 5.2.

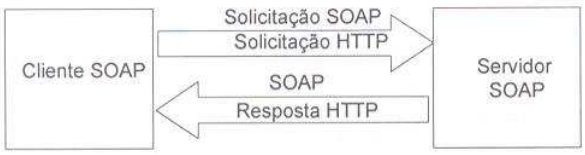

Figura 5.2 Fluxo de Comunicação com Uso do SOAP

- Atravessa *firewalls* e roteadores, que o enxergam com uma comunicação HTTP.
- Tanto os dados como as funções são descritas em XML, o que torna o protocolo não apenas fácil de usar como também muito robusto.
- É independente do sistema operacional e linguagem de programação.
- Pode ser usado tanto de forma anônima como com autenticação (nome/senha).
- O SOAP é razoavelmente superficial como um protocolo.
- Existe suporte para SOAP, por parte de vários fornecedores, incluindo a Microsoft, a IBM e a SUN.

A analogia que se faz a este protocolo em face de uma arquitetura utilizando-se de J2EE ou mesmo CORBA são os protocolos RMI (*Remote Method Invocation*) ou IIOP (*Internet Inter-OP*). Notadamente, o SOAP possui muitas vantagens em relação a tais protocolos, dentre elas destacam-se:

- Não se utiliza de mecanismos de seleção dinâmica de *ports* e, portanto, pode processar facilmente através de *firewalls* padrões.
- Pelo fato de ser derivado do padrão XML pode ser utilizado para dispor argumentos de qualquer aplicação para qualquer outra aplicação, não importando o tipo de sistema, linguagem de programação ou sistema distribuído.

O SOAP também pode ser usado com outros protocolos, além do HTTP, como o SMTP e o *Java Messaging Server* (JMS). Entretanto, o HTTP é o protocolo utilizado com mais freqüência, pois é amplamente aceito. Além disso, o modelo de mensagens SOAP mapeia naturalmente para o modelo de solicitação/resposta http. W3C (16)

5.3 Mensagens SOAP com Anexos

SOAP *Messages with Attachments* (Mensagens SOAP com anexos) é uma especificação que define um mecanismo para o envio de um ou mais anexos na mensagem SOAP. Um anexo pode ser representado por qualquer tipo de dados binários, como imagens de fac-símile de documentos legais, ou dados não-binários, como esboços de engenharia. A especificação não define nenhum novo mecanismo de codificação. Pelo contrário, emprega o padrão MIME, que é um mecanismo de codificação usado para o envio de anexos de e-mail. A especificação utiliza os mecanismos do MIME para empacotar uma mensagem SOAP e fazer referência aos seus anexos.

5.4 Objetivos do Projeto SOAP

Os principais objetivos do projeto da especificação SOAP 1.1. eram a simplicidade e a extensibilidade. O SOAP satisfazia completamente o objetivo do projeto relativo à simplicidade, fazendo do protocolo um mecanismo superficial simples para a troca de informações estruturadas e antecipadas entre sistemas distribuídos. O SOAP não possui recursos que estejam presentes em sistemas de mensagens tradicionais e de objetos distribuídos, como:

- Coleta de lixo distribuída
- Lotes de mensagens
- Objeto-por-referência
- Ativação

Os detalhes sobre a coleta de lixo são deixados para cada implementação do SOAP. O SOAP também não aceita lotes de mensagens. Os lotes de mensagens representam a capacidade de enviar várias mensagens durante uma transmissão, em vez de uma única mensagem, o que minimiza o tráfego de rede e melhora a performance, pois é minimizado o número de idas e vindas entre o cliente e o servidor. Entretanto, esse suporte implica na criação de algumas regras sofisticadas para a manipulação das mensagens ao serem recebidas, como quais mensagens deveriam ser processadas e o que aconteceria se uma delas não pudesse ser processada. A complexidade desse recurso entraria em conflito com a simplicidade do objetivo do projeto.

O CORBA utiliza objeto por referência para permitir que objetos do servidor remoto sejam manipulados pelos clientes. Isso é feito através dos *stubs*. Portanto, para

manipular um objeto remoto, o *stub* deve ser obtido, o que pode não ser muito prático. O SOAP resolve esse problema, permitindo que os objetos sejam manipulados usando o protocolo XML, em vez de objeto por referência, o que facilita a manipulação de objetos provenientes de praticamente qualquer plataforma.

A ativação é a capacidade de especificar como o objeto do servidor remoto é chamado. A ativação também não é suportada pelo SOAP. W3C (16)

5.5 Aspectos de Segurança

Diferentemente de outros protocolos de computação distribuídos, o SOAP pode utilizar o HTTP como um protocolo de transporte. Devido à maioria dos *firewalls* de empresas permitir o tráfego no HTTP, tem-se que considerar um novo leque de riscos relacionados à segurança. A especificação SOAP lida com esses problemas, definindo um novo campo de cabeçalho de solicitação HTTP, denominado SOAPAction. Um cliente SOAP que utilize o transporte no HTTP deve usar esse campo, quando emitir uma solicitação SOAP.

O valor do campo deveria ser um *Uniform Resource Identifier* (URI) que identifica a intenção da mensagem SOAP. Isso poderia ser usado por *firewalls* para filtrar, adequadamente, as mensagens de solicitação SOAP.

5.6 A Estrutura do SOAP

O SOAP utiliza a XML para codificar todas as mensagens. Tendo em vista a simplicidade, as mensagens SOAP não podem conter um DTD. Uma mensagem SOAP deve incluir os *namespaces* característicos da XML em todos os elementos e atributos definidos pelo SOAP.

Se os *namespaces* não forem especificados, a aplicação SOAP talvez não processe a mensagem SOAP. A vantagem de usar *namespaces* no SOAP é que os elementos definidos não entrarão em conflito com os elementos do SOAP padrão.

Para a codificação de dados via XML Schemas usa-se o *namespaces*

xmlns:xsi=”http://www.w3.org/2001/X MLSchema-instance”

xmlns:xsd=“http://www.w3.org/2001/XMLSchema“

O uso de *namespaces* faz com que a aplicação tenha modularidade, enquanto também provê flexibilidade para mudanças futuras às aplicações. W3C (16)

5.6.1 Mensagem SOAP

Uma mensagem SOAP é um documento XML que pode conter, no máximo, três partes: O envelope SOAP (obrigatório), o cabeçalho SOAP (opcional) e o corpo SOAP (obrigatório).

A figura 5.3 exibe uma representação gráfica de uma mensagem SOAP:

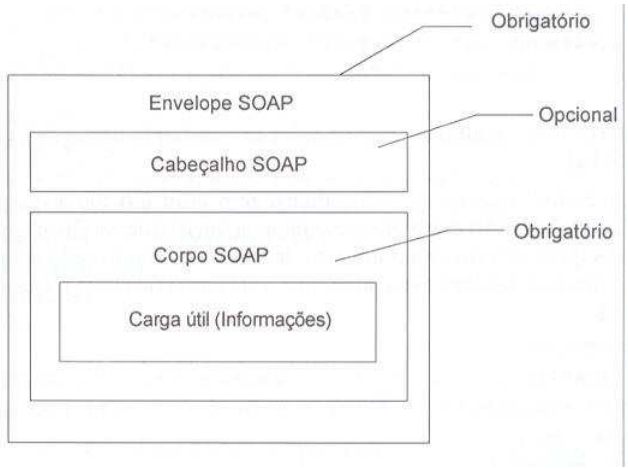

Figura 5.3 Estrutura de uma Mensagem SOAP

5.6.1.1 Envelope SOAP

O envelope SOAP XML define regras específicas para encapsular dados sendo transferidos entre computadores. Isto inclui dados específicos da aplicação, tais como nomes de métodos a invocar, parâmetros de métodos, ou valores de retorno. Pode também incluir informação sobre quem deve processar os conteúdos de envelopes, e no caso de falha, como codificar mensagens de erro.

O envelope SOAP é o elemento de nível mais elevado de uma mensagem SOAP. Pode ser interpretado como o equivalente a um envelope usado para enviar cartas através do serviço de correios convencional. Contém um cabeçalho SOAP opcional e um corpo SOAP obrigatório. O envelope SOAP é designado em uma mensagem SOAP pelo elemento <Envelope>.

Todos os elementos recebem o prefixo identificador de *namespace* SOAP-ENV, que está associado ao *namespace* http://schemas.xmlsoap.org/soap/envelope. O objetivo dessa definição é garantir que os elementos SOAP não entrem em conflito com elementos que possam ser definidos.

Uma aplicação SOAP que receba uma mensagem com um *namespace* diferente de http://schemas.xmlsoap.org/soap/envelope deve retomar um erro de versão, negligenciando a mensagem. O envio de mensagens de erro como um erro de versão é feito usando-se o elemento <*Fault*> do SOAP.

Em suma as regras gramaticais a serem obedecidas em um envelope são:

a. O nome do elemento é “Envelope”

b. O elemento deve ser representado em uma mensagem SOAP

O elemento pode conter declarações *namespace* bem como atributos adicionais. Se presente, tal atributo adicional deve ser um *namespace* qualificado. Similarmente, o elemento pode conter sub-elementos adicionais. Se presente estes elementos também devem ser *namespace* qualificados e devem seguir o elemento corpo SOAP.

O atributo global *encodingStyle* (estilo de codificação) pode ser usado para indicar as regras de codificação do documento usadas na mensagem SOAP. Esse atributo pode aparecer em qualquer elemento, e é definido para que os conteúdos dos elementos e todos os elementos filhos não precisem conter eles mesmos tal atributo, embora uma declaração *namespace* XML seja necessária. Não há codificação padrão definida para uma mensagem SOAP. W3C (16)

5.6.1.2 Cabeçalho SOAP

O elemento *Header* (cabeçalho SOAP) é parte opcional de uma mensagem SOAP, que oferece uma estrutura flexível para especificar requisitos adicionais no nível da aplicação.

O cabeçalho SOAP é uma maneira flexível e geral de adicionar recursos como autenticação, gerenciamento de transação e serviços de pagamento a uma mensagem SOAP. Um cabeçalho SOAP é especificado em uma mensagem SOAP, pelo elemento SOAP-ENV:Header. Um cabeçalho SOAP não é obrigatório, mas, se for usado, deverá vir imediatamente depois do elemento envelope SOAP, podendo conter elementos filhos, que são representados como entradas de cabeçalho dentro da especificação SOAP.

As informações de autenticação são especificadas como uma entrada de cabeçalho. Todas as entradas de cabeçalho de primeiro nível devem ser plenamente qualificadas, o que significa que cada elemento deve estar associado a um *namespace*. Isso não é necessário para elementos filhos de entradas de cabeçalho.

As entradas do cabeçalho SOAP podem conter dois atributos: *actor* e *mustUnderstand*. O atributo *actor* pode ser usado para indicar quem deve processar um elemento cabeçalho. O atributo *mustUnderstand* pode ser usado para indicar se uma entrada de cabeçalho é obrigatória, para que a mensagem SOAP seja processada. Se a aplicação SOAP não souber como processar a entrada do cabeçalho, o processamento da mensagem poderá falhar. Caso contrário, a mensagem deverá ser processada.

Uma aplicação SOAP deve ignorar todos os atributos de cabeçalho SOAP que não sejam aplicados a um filho imediato de um elemento de cabeçalho SOAP.

Na definição do cabeçalho SOAP os *details* do elemento *Header* são finalizados abertos, provendo assim, flexibilidade máxima para desenvolvedores de aplicação.

A regra gramatical para elaboração do cabeçalho versa o seguinte:

a. O nome do elemento é "*Header*".

b. O elemento pode estar presente em uma mensagem SOAP. Se presente o elemento deve ser o primeiro elemento filho imediato de um elemento de envelope SOAP.

c. O elemento pode conter um conjunto de entradas *Header* cada uma sendo um elemento filho imediato do elemento *Header* SOAP. Todos os elementos filhos imediatos do elemento *Header* SOAP devem ter um *namespace* qualificado.

O atributo *actor* pode ser usado para indicar quem deve processar um elemento cabeçalho. O valor do atributo *actor* é baseado na hipótese de que cada processador de SOAP é associado a um URI.

O receptor de um elemento de cabeçalho SOAP não pode encaminhar o elemento de cabeçalho ao próximo intermediário. A especificação SOAP indica que o papel de um receptor de um elemento cabeçalho é semelhante ao da aceitação de um contrato, à medida que não pode ser estendido para além do receptor. Entretanto, o receptor pode inserir uma entrada de cabeçalho semelhante, mas agora o contrato está entre a aplicação e o receptor do *actor* do cabeçalho.

Um atributo *actor* também pode conter um valor para URI igual a http://schemas.xmlsoap.org/soap/actor/next. Esse URI indica que o elemento do cabeçalho está destinado à primeira aplicação que processe a mensagem.

O atributo global SOAP *mustUnderstand* pode ser usado para indicar se uma entrada de cabeçalho é obrigatória ou opcional para um destinatário processar. O destinatário de uma entrada de cabeçalho é definido pelo atributo *actor* SOAP. O atributo SOAP *mustUnderstand* pode assumir os valores “1” ou “0”. A ausência do atributo SOAP *mustUnderstand* é semanticamente equivalente a sua presença com o valor “0”.

Se o elemento de cabeçalho é rotulado com o atributo SOAP *mustUnderstand* com o valor de “1”, o destinatário daquela entrada de cabeçalho também deve obedecer às semânticas (como transportado pelo nome qualificado do elemento) e processar corretamente para aquelas semânticas, caso contrário o processamento da mensagem falha. Um valor 0 indica que o receptor não precisa compreender o elemento para poder processar a mensagem. W3C (16)

5.6.1.3 Corpo SOAP

O corpo SOAP contém as informações que devem ser recebidas pelo receptor da mensagem. O corpo SOAP de uma mensagem é especificado usando o elemento <*Body*>. Na prática, essas informações, consistem, tipicamente, em uma chamada RPC, resposta RPC, ou relatório de erro. Porém, teoricamente, um corpo SOAP pode conter qualquer tipo de informação. Caso um elemento de cabeçalho esteja presente, o elemento <*Body*> deve seguir imediatamente o elemento <*Header*>.

Da mesma maneira que um elemento, todos os elementos filhos imediatos do elemento do SOAP são denominados entradas do corpo. Uma entrada do corpo é identificada por um nome de elemento plenamente qualificado. Os elementos filhos imediatos de uma entrada de corpo não precisam ser qualificados por *namespace*.

Em resumo a sintaxe para construção do elemento corpo SOAP deve obedecer as seguintes regras:

a. O nome do elemento é “*Body*”.

b. O elemento deve estar presente em uma mensagem SOAP e deve ser um elemento filho intermediário de um elemento envelope SOAP. Ele deve diretamente seguir o elemento *Header* SOAP se presente. Caso contrário ele deve ser o primeiro elemento filho imediato do elemento envelope SOAP.

c. O elemento pode conter um conjunto de entradas *Body*, cada uma sendo um elemento filho imediato do elemento *Body* SOAP. Elementos filhos imediatos

do elemento *Body* SOAP podem ser um *namespace* qualificado. SOAP define o elemento *fault* SOAP, que é usado para indicar mensagens de erros.

O elemento <*Fault*> do SOAP é usado para o tratamento de erros em uma aplicação SOAP. O elemento <*Fault*> pode ocorrer uma única vez dentro de uma mensagem SOAP. Não se pode ter vários elementos <*Fault*> do SOAP. Vale a pena notar que o elemento <*Fault*> do SOAP é a única entrada de corpo definida na especificação SOAP.

O elemento *fault* SOAP define quatro subelementos:

- Faultcode - Código de falha: O elemento *faultcode* é dirigido para o uso de um software para prover um mecanismo algorítmico a fim de identificar a falha. SOAP define um pequeno conjunto de códigos de falha cobrindo as falhas básicas SOAP.

- Faultstring - *String* de falha: O elemento *faultstring* destina-se a fornecer uma explicação que pode ser lida pelo humano e não é dirigido para processamento algorítmico. Ele deve estar presente em um elemento *fault* SOAP e deveria prover pelo menos alguma informação explicando a natureza da falha.

- Faultactor - Ator de falha: O elemento *faultactor* é dirigido a prover informação sobre quem levou a falha a acontecer dentro do caminho de mensagem. Ele é similar ao atributo *actor* SOAP, mas ao invés de indicar o destino da entrada de cabeçalho, ele indica a fonte da falha. O valor do atributo *faultactor* é uma URI identificando a fonte.

- Detail – Detalhe: O elemento *detail* é dirigido para carregar aplicação específica relacionada à informação de erro para o elemento *Body*. Ele deve estar presente se os conteúdos do elemento *Body* não puderem ser processados com sucesso. A ausência do elemento *detail* no elemento *fault* indica que a falha não é relacionada ao processamento do elemento *Body*. Isso pode ser usado para distinguir se o elemento *Body* foi processado ou não em caso de uma situação de falha.

Outros subelementos de falha podem estar presentes, contanto que eles sejam *namespace* qualificados. W3C (16)

5.6.2 Codificação do SOAP

A especificação SOAP 1.1 define como os dados na carga útil da mensagem SOAP podem ser codificadas. Entretanto, o SOAP não define um novo sistema de tipos, mas o adota a partir da especificação "XML Schema Part 2: Datatypes" (http://www.w3.org/TR/xmlschema-2/).

A especificação define dois grupos de tipos: tipos simples e tipos complexos. A diferença entre tipos simples e tipos complexos é que os tipos simples não podem ter filhos de elementos ou atributos, enquanto os tipos complexos podem ter filhos de elementos e atributos. W3C (16)

5.6.2.1 Tipos simples

A especificação "XML Schema Part 2: Datatypes" define 44 tipos simples incorporados que podem ser especificados usando o atributo *xsi-type* em um elemento.

Os tipos de dados declarados na especificação esquema XML podem ser usados diretamente em esquemas de elementos. Os Tipos derivados desses podem também ser usados.

5.6.2.2 Tipos compostos

O SOAP define dois tipos compostos: *struct* e *array*. O tipo composto *struct* permite que valores de tipos diferentes sejam agrupados, sendo cada valor armazenado e recebido usando um único nome de assessor. O tipo composto *array* consiste em valores que são armazenados e recuperados usando um número ordinal.

O tipo composto *struct* é bem parecido com uma estrutura dentro da linguagem de programação C. Da mesma maneira que os tipos simples, os tipos compostos podem ser multi-referência.

Os *arrays* podem ser definidos como tendo um tipo SOAP-ENC:Array ou um tipo derivado dele. Os elementos contidos dentro de um *array* podem ser de qualquer tipo, incluindo *arrays* aninhados.

Os tipos de dados *struct*, ou outros valores compostos, podem representar valores dentro de um *array*. Os *arrays* podem ainda ter outros *arrays* como valores de membro. W3C (16)

5.7 SOAP em HTTP

Teoricamente, uma mensagem SOAP pode ser vinculada a qualquer protocolo de transporte, mas o HTTP definido na especificação SOAP como o mais amplamente usado. O SOAP se ajusta satisfatoriamente ao modelo de mensagem de solicitação/resposta HTTP, o que permite que uma solicitação SOAP seja transmitida por uma mensagem de solicitação HTTP, e que a resposta SOAP seja transmitida por uma mensagem de resposta HTTP, conforme pode ser visto na figura 5.2.

5.7.1 Solicitação HTTP do SOAP

A solicitação HTTP do SOAP é uma solicitação HTTP que contém uma mensagem SOAP. A mensagem SOAP contém, por sua vez, uma solicitação SOAP (por exemplo, uma chamada RPC). A solicitação HTTP do SOAP deve ser transmitida usando o método POST do HTTP. O campo de cabeçalho *SOAPAction* deve ser usado para indicar a intenção da solicitação HTTP do SOAP.

```
POST rpcrouter HTTP/1.1

Host: 127.0.0.1

Content-Type: textlxml; charset="utf-8"

Content-Length: 559

SOAPAction:

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<SOAP-ENV:Header>

<a:authentication xmlns:a="http://www.wrox.com/soap/authentication"

SOAP-ENV:mustUnderstand="1">

<a:username>mlhendri</a:username>

<a:password>courtney</a:password>
```

```
</a:authentication>

</SOAP-ENV:Header>

<SOAP-ENV:Body>

<cmd:processReboot xmlns:cmd="http://www.wrox.com/soap/cmd">

<ip xsi:type="xsd:string">192.168.1.3</ip>

<delay xsi:type="xsd:int">30000</delay>

<cmd:processReboot>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope> W3C (16)
```

5.7.2 Resposta HTTP do SOAP

Nas mensagens SOAP que contém uma mensagem de resposta SOAP os códigos de status do HTTP são usados para comunicar informações de status no HTTP. Um código de status de 2xx indica que a solicitação do cliente, incluindo o componente do SOAP, foi recebida e compreendida, com sucesso. O trecho de código a seguir descreve isso:

```
HTTP/1.1 200 OK

Content-Type: textlxml; charset="utf-8"

Content-Length: 320

<SOAP-ENV:Envelope>

</SOAP-ENV:Envelope>
```

Se um erro do SOAP ocorrer durante o processamento da solicitação, o servidor HTTP deverá emitir uma resposta HTTP 500 "*Internal Server Error*", que também deve incluir uma resposta que contenha um elemento <*Fault*> do SOAP, indicando o erro do processamento. W3C (16)

5.7.3 RPC no SOAP

Uma das principais vantagens proporcionadas pela utilização do SOAP é a capacidade de encapsular chamadas RPC dentro da carga útil de uma mensagem SOAP. Conforme já foi mencionado, uma mensagem SOAP pode ser vinculada, potencialmente, a qualquer protocolo de transporte. Entretanto, uma chamada RPC é mapeada naturalmente para uma solicitação HTTP, enquanto uma resposta RPC é mapeada naturalmente para uma resposta HTTP. Para criar um método usando o HTTP, ou qualquer outro protocolo de transporte, são necessárias as seguintes informações:

- O URI de solicitação do objeto de destino
- Dados de cabeçalho (opcional)
- O nome de um método
- A assinatura de um método (opcional)
- Os parâmetros do método

O URI de solicitação do objeto de destino não é tratado pelo SOAP, mas deve ser fornecido pelo protocolo de transporte. Caso o HTTP seja usado como o protocolo de transporte, o URI de solicitação será usado para indicar ao recurso contra o qual está sendo feito o chamamento. O SOAP não impõe nenhuma restrição na forma do URI, com a exceção de que o URI deve ser válido. W3C (16)

5.8 Normalização do SOAP

Este capítulo foi baseado na sua grande maioria na documentação da norma SOAP, disponível na Internet. A documentação, bem como o esquema e demais informações sobre o SOAP podem ser obtidas através do site da W3C em http://www.w3c.org.

5.9 Relação a XML

Todas as mensagens SOAP são codificadas usando XML.

Para descrever a codificação, a seguinte terminologia é usada:

1. Um “valor” é uma *string*, o nome de uma medida (número, data, enumeração) ou uma composição de vários valores primitivos. Todos os valores são de tipos específicos.

2. Um “valor simples” é sem partes nomeadas. Exemplos de valores simples são *strings* particulares, *integers*, valores enumeráveis.

3. Um “valor composto” é uma agregação de relação de outros valores. Exemplos de valores compostos são ordem de compra particular, relatórios de estoque, endereços de ruas, entre outros.

4. Dentro de um valor composto, cada valor relacionado é potencialmente distingüível por um nome, ordinal ou ambos. Isso é chamado seu “assessor”. Exemplos de valores compostos incluem ordens de compras particulares, relatórios de estoques. Vetores são também valores compostos. É possível ter valores compostos com vários assessores cada um nomeado igual.

5. Um “vetor” é um valor composto no qual a posição ordinal serve como a única distinção entre valores membros.

6. Uma “estrutura” é um valor composto no qual o nome assessor é a única distinção entre valores membros, e cada assessor possui um nome exclusivo.

7. Um “tipo simples” é uma classe de valores simples. Exemplos de tipos simples são as classes chamadas “*string*”, “*integer*”, classes de enumeração.

8. Um “tipo composto” é uma classe de valores compostos. Um exemplo de tipo composto é a classe de valores de ordem de compra compartilhando o mesmo assessor embora com valores potencialmente diferentes.

9. Dentro de um tipo composto, se um assessor tem um nome que é distinto dentro daquele tipo, mas não é distinto com relação a outros tipos, que é o nome mais o tipo juntos é necessário fazer uma identificação única, o nome é definido localmente. Se, entretanto o nome é baseado em partes em um identificador de recursos uniformes, diretamente ou indiretamente, o nome somente é suficiente para identificar unicamente o assessor do tipo dentro do qual ele aparece, o nome é chamado “universalmente definido”.

10. Dada a informação no esquema relativo para qual um gráfico de valores é serializado, é possível determinar que alguns valores podem somente ser

relacionados por uma instância simples de um assessor. Para outros, não é possível fazer essa determinação. Se somente um assessor pode referenciá-lo, um valor é considerado “referência simples”. Se ele é referenciado por mais que um ele é “multi-referência”. Isso é possível por um certo valor ser considerado “referência simples” relativo a um esquema e “multi-referência“ relativo a um outro.

11. Sintaticamente, um elemento pode ser “independente” ou “embutido”. Um elemento independente é qualquer elemento aparecendo no nível de topo da codificação do documento. Todos os outros são elementos embutidos.

Algumas regras de codificação do documento são:

1. Todos os valores são representados como conteúdo de elemento. Um valor multi-referência deve ser representado como o conteúdo de um elemento independente.

2. Um valor simples é representado com um dado caractere, sem quaisquer subelementos. Todo valor simples deve ter um tipo que é também listado na especificação de esquemas XML

3. Um valor composto é codificado como uma seqüência de elementos, cada assessor representado por um elemento embutido cujo nome corresponde ao nome do assessor. Assessores cujos nomes são locais ao seus tipos de nomes de elementos não qualificados; todos os outros têm nomes qualificados.

4. Vetores SOAP tem uma ou mais dimensões cujos membros são distinguíveis pela posição ordinal. Um valor vetor é representado por uma série de elementos refletindo o vetor, com membros aparecendo em seqüência ordinal ascendente. Para vetores multidimensionais a dimensão no lado direito varia mais rapidamente. Cada elemento membro é nomeado com um elemento independente.

6 WSDL

A WSDL (W*eb Service Description Language*) foi definida em um esforço conjunto da IBM, Microsoft e Ariba, tendo sido padronizada pela W3C e constitui-se numa tecnologia baseada em XML para definir interfaces para serviços web, dados e tipos de mensagens, modelos de interação (*interaction patterns*) e mapeamento de protocolos (*protocol mappings*).

Assumindo que um dos objetivos do serviço web é permitir que aplicações desenvolvidas em diferentes linguagens ou mesmo modelos de programação diversos possam operar entre si, o mesmo utiliza-se da tecnologia WSDL para tanto.

A analogia que se faz a esta linguagem em face de uma arquitetura utilizando-se de J2EE ou mesmo CORBA são os protocolos RMI ou a linguagem IDL (*Interface Definition Language*).

A notação que o arquivo WSDL usa para descrever o formato das mensagens é baseado no padrão XML, o que significa que é uma linguagem de programação neutra e baseada em padrões, o que a torna adequada para descrever as interfaces dos serviços web, que são acessíveis por uma grande variedade de plataformas e linguagens de programação. Além de descrever o conteúdo das mensagens, o WSDL define onde o serviço está disponível e quais protocolos de comunicação são usados para conversar com o serviço. Isso significa que o arquivo WSDL define tudo que é necessário para escrever um programa que utilize o XML serviço web. Há várias ferramentas disponíveis para ler o arquivo WSDL e gerar o código para comunicar com o XML serviço web, sendo que a grande maioria delas foram desenvolvidas com o uso de API´s ou recursos da linguagem Java.

6.1 Namespaces no WSDL

Cada documento WSDL é um documento XML e, portanto, os *namespaces* são definidos em seu interior. Para maximizar a taxa de reutilização dos componentes de um

documento WSDL, muitos dos elementos na WSDL contêm atributos para fazer referência a outros elementos, seja dentro ou fora do documento. O elemento <*binding*>, por exemplo, define um atributo que indica um elemento <*portType*> .

O *namespace* padrão que cada documento WSDL deve utilizar é http://schemas.xmlsoap.org/wsdl/.

O Quadro 6.1 apresentada uma lista de *namespaces* que se utiliza com regularidade, em documentos WSDL, junto com seus prefixos recomendados:

Prefixo	URL do namespace	Descrição
WSDL	http://schemas.xmlsoap.org/wsdl/	Namespace de WSDL para framework WSDL
Soap	http://schemas.xmlsoap.org/wsdl/soap/	Namespace WSDL para vínculo de SOAP de WSDL
http	http://schemas.xmlsoap.org/wsdl/http	Namespace de WSDL para WSDL HTTP GET & vínculo POST
Mime	http://schemas.xmlsoap.org/wsdl/mime	Namespace WSDL para vínculo MIME de WSDL
Soapenc	http://schemas.xmlsoap.org/soap/encoding	Namespace de codificação conforme definido pelo SOAP 1.1
Soapenv	http://schemas.xmlsoap.org/soap/envelope	Namespace do envelope conforme definido pelo SOAP 1.1
Xsi	http://www.w3.org/2001/XMLSchema-instance	Namespace da instância conforme definido pelo esquema de XML
Xsd	http://www.w3.org/2001/XMLSchema	Namespace do esquema; conforme definido pelo esquema de XML

Quadro 6.1 Lista de Namespaces usados em WSDL

6.2 Estrutura do Documento WSDL

Um documento WSDL representa a interface externa de um serviço web. Entretanto, além de descrever a interface apresentada, um documento WSDL também contém a localização (ou 'extremidade' como a especificação se refere a ela) do serviço. O registro do serviço está disponível em um local previsível e bem conhecido, na rede, o que permite que o cliente contate-o de uma maneira confiável. Isso significa, sem dúvida, um beneficio adicional - o da independência de local. Os serviços podem ser movidos de acordo com considerações comerciais ou de distribuição, de uma maneira que elimine a necessidade de que os clientes sejam informados sobre as alterações realizadas.

A programação de extremidade também pode ser comparada à programação de socket: uma aplicação que vincula um socket requer o conhecimento do endereço ao qual vincular, e o protocolo da aplicação que opera no *socket*, ou seja, a lista de mensagens de entrada aceitáveis e dos formatos de mensagens de saída esperados.

A combinação das interfaces com a extremidade disponibiliza o serviço para um cliente.

Para conferir a maior flexibilidade possível, a WSDL define os vários componentes de um serviço que podem ser, então, reutilizados para definir diferentes serviços. Os componentes que compõem um serviço incluem:

- Tipos de dados: (*String, int, Object*, entre outros). Os tipos de dados na WSDL são conhecidos como tipos.
- Parâmetros de entrada e saída de um serviço: Um parâmetro de chamada de um serviço é denominado mensagem.
- O relacionamento entre parâmetros de entrada e saída: A assinatura do método, também conhecida como operações.
- Agrupamento lógico de operações: Isso pode ser equacionado com um determinado objeto, que é um agrupamento lógico de métodos Esse agrupamento lógico é conhecido como um Tipo de Porta.
- O protocolo a ser usado para acessar os métodos de um objeto: As únicas opções definidas atualmente na especificação WSDL são SOAP, HTTP e MIME.
- Endereço do serviço: O endereço de um serviço, além dos componentes acima, define o serviço.

Os quatro primeiros são similares à sistemática utilizada em Java, onde freqüentemente é necessário saber qual o objeto está acessando o método, e seus parâmetros de entrada e saída. Geralmente, não é necessário conhecer todos os métodos expostos por um objeto, embora seja desejável. Os dois últimos itens na lista acima são necessários para aplicações distribuídas. Embora o SOAP e o HTTP sejam, atualmente, os únicos protocolos aceitos, isso não significa necessariamente que eles não sejam substituídos ou complementados por outros, a curto ou médio prazo. O endereço do serviço é necessário, descrevendo meramente a localização do serviço na rede.

O mapeamento, pela estrutura de um documento WSDL, é feito da seguinte maneira:

- O elemento <*types*> define os tipos de dados usados.
- Os elementos <*message*> definem as mensagens usadas pelo serviço.
- Os elementos <*operation*> definem mensagens solicitação-resposta, usadas pelas funções individuais oferecidas por um serviço.
- Um elemento <*portType*> encapsula uma coleção de operações.

- O elemento <*binding*> descreve como um tipo de porta é mapeado para um protocolo de chamada de rede, como o SOAP.
- O elemento <*service*> e seu elemento <*port*> contido incluem a localização da implementação de um serviço na rede.

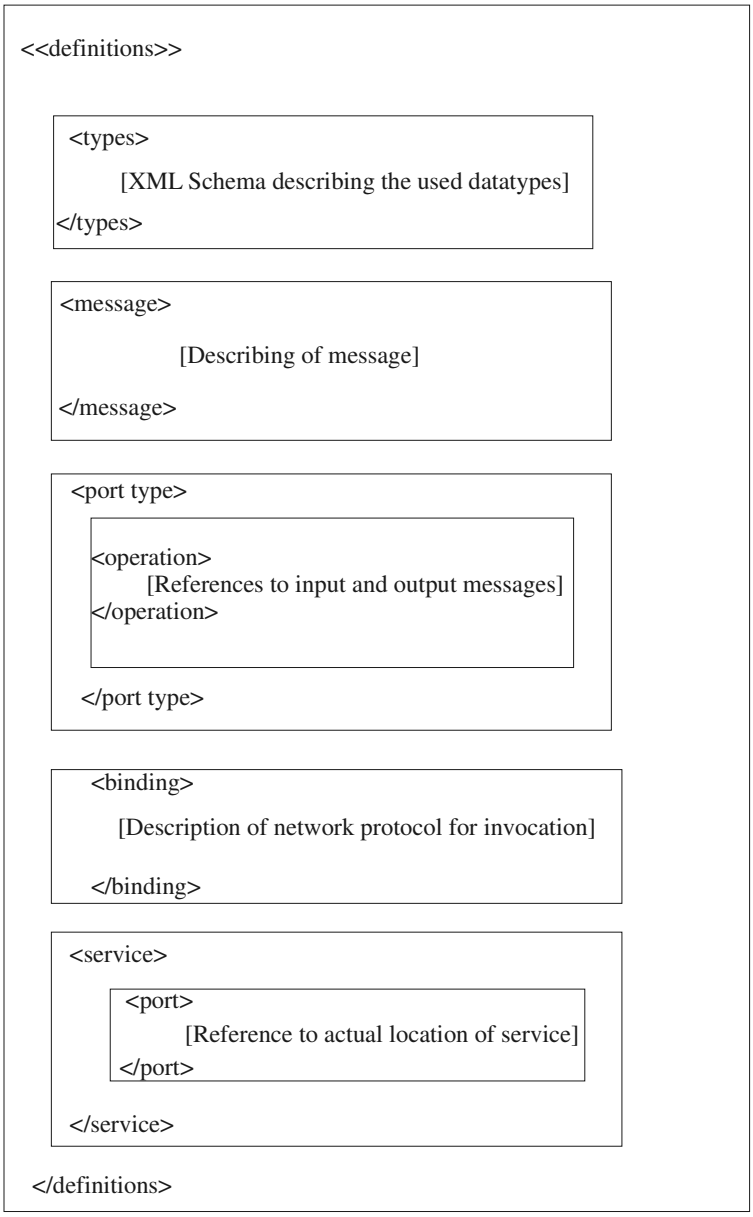

Figura 6.1 Estrutura de um Documento WSDL

O diagrama apresentado na figura 6.1 resume os principais elementos da WSDL que podem ocorrer em um documento. W3C (16)

6.2.1 O Elemento <definitions>

O elemento raiz de cada documento WSDL é o elemento <*definitions*>, que contém tipicamente atributos definindo os *namespaces* usados no documento WSDL. W3C (16)

6.2.2 O elemento <import>

Um documento WSDL não está necessariamente contido em um arquivo físico e pode ser dividido em vários arquivos. Por exemplo, informações sobre interface ou tipos de dados podem ser mantidas separadas das definições de extremidade concretas, o que permite a reutilização das definições de interface potencialmente padronizada a partir das implementações reais em um servidor que está sendo executado. Além do mais, as definições de esquema, conforme contidas no elemento <*types*> , podem ser importadas de locais separados. A maioria das ferramentas e plataformas deve aceitar algum tipo de separação entre as várias partes de uma definição de WSDL.

A WSDL contém um elemento denominado <*import*>, que aceita essa separação. Ele contém dois atributos, um definindo a localização do documento importado, e outro definindo seu *namespace*. W3C (16)

6.2.3 O elemento <types>

Qualquer serviço web representativo lida com dados. Os dados devem ser passados ao serviço, retomados do serviço, ou ambos. Em Java utiliza-se objetos e tipos primitivos para a troca de dados entre diferentes partes das nossas aplicações. Em aplicações distribuídas, esses objetos são passados, tipicamente, por uma rede, no formato binário. Em outras palavras, eles são ordenados ou serializados. Ambos os lados do caminho de comunicação sabem como serializar e desserializar os dados para objetos.

No ambiente de serviços web, o princípio é o mesmo, exceto pelo fato de a serialização dos objetos ocorrer na XML, para obter vantagem dos benefícios da utilização de dados neutros da plataforma.

O elemento <*types*> contém a definição dos tipos de dados com os quais um serviço lida. Em quase todos os casos, esta e uma definição de XML. Caso ela não esteja presente, isso significa que um serviço esteja usando apenas tipos de dados básicos, como *strings*, inteiros e outros. W3C (16)

6.2.4 O elemento <message>

Uma mensagem é uma unidade conceitual de dados que podem ser trocados. Uma mensagem pode ser composta de alguns argumentos no caso da chamada de um método todos os parâmetros do método são representados por uma única mensagem. Para esta finalidade, cada elemento <*message*> contém um ou mais elementos <*part*> que formam as partes reais da mensagem. Cada elemento <*part*> se refere a um tipo definido no elemento <*types*> do documento. W3C (16)

6.2.5 Os elementos <operation> e <portType>

A próxima etapa na definição de um serviço web consiste em obter as mensagens conforme declaradas pelos elementos <*message*> em uma seqüência útil. A especificação WSDL chama isso de operação, que é representada por uma ou mais mensagens. Existem quatro tipos de mensagens diferentes de operações:

- Operação unidirecional: Este tipo de operação define uma mensagem que é enviada de um cliente para um serviço, sem que ocorra nenhuma resposta desse serviço.
- Solicitação-Resposta: Este é o tipo mais comum de operação. Um cliente envia uma solicitação a um serviço e recebe uma mensagem de resposta, como resultado dessa solicitação.
- Pedido-Resposta: Uma operação de pedido-resposta define uma mensagem enviada do serviço para o cliente. Isso resulta em uma mensagem, que é enviada do cliente de volta ao serviço. Essa não é uma utilização muito freqüente dos serviços web.
- Notificação: O serviço envia uma mensagem ao cliente, correspondendo, talvez, à resposta de uma assinatura feita pelo cliente.

A operação do tipo solicitação-resposta pode ser vista apenas como uma RPC (*remote procedure call*). A chamada de procedimento remoto é uma chamada de função pelos limites de processo que utilizam alguns parâmetros e retomam um resultado. A operação solicitação-resposta é semelhante. A solicitação por um serviço resulta no retomo de uma resposta. Essa operação consiste em uma mensagem de entrada e de uma mensagem de saída. As mensagens que são usadas por este tipo de operação devem ser definidas pelos elementos <*message*> no documento WSDL.

Na WSDL, uma ou mais operações podem ser agrupadas em um elemento <*portType*>. Uma classe java possui qualquer quantidade de métodos que podem ser

chamados nas suas instâncias. Analogamente, um tipo de porta de serviço web define qualquer quantidade de operações. Entretanto, um <*portType*> corresponde apenas a um agrupamento conceitual de operações. É possível agrupar todas as operações em um documento WSDL em um ou vários elementos <*portType*> essa é uma decisão relacionada ao projeto, bem como o mapeamento de todos os métodos de uma classe para um tipo de porta. W3C (16)

6.2.6 O elemento <binding>

O elemento <*binding*> descreve o mecanismo usado por um serviço, para se comunicar com um cliente. Esse elemento é mais complexo do que os elementos anteriores, pois ele tem um certo grau de liberdade, quanto ao que se pode incluir nele. A WSDL foi escrita para que seja independente do mecanismo efetivo usado para acessar um serviço. O SOAP é a opção dominante para este acesso, mas outros mecanismos também são utilizados. Além do SOAP e do HTTP, a especificação descreve um vínculo em MIME. Isso pode ser usado para descrever como uma determinada mensagem é codificada, com relação ao seu tipo MIME. W3C (16)

6.2.7 Os elementos <service> e <port>

Esses elementos definem para onde enviar a solicitação real. Para ser exato, o elemento <*port*> no elemento <*service*> contém essa informação. Diversas portas podem ser especificadas. Cada porta é específica para o tipo de vínculo que foi descrito no elemento <*binding*>. Em outras palavras, a definição da porta de um vínculo SOAP é diferente daquela para outros vínculos. Isso também permite que um serviço web seja acessível através de várias portas, no mesmo instante. W3C (16)

6.3 WSDL e Java

No ambiente de serviços web, pode-se assumir o papel de um provedor de serviço, um solicitante ou ambos. A utilização de WSDL diferirá, dependendo do papel que se desempenha.

No caso de um provedor que queira implementar o seu serviço em Java, existem duas abordagens possíveis: *top-down* ou a *bottom-up*.

No caso da abordagem *top-down*, já existe uma definição de WSDL para um serviço. Por exemplo, uma definição de WSDL padronizada pode permitir a recuperação de uma entrada em um catálogo de endereços. Caso se queira fornecer a implementação de um serviço para um catálogo de endereços, e seguir o padrão, é necessário traduzir as definições do tipo de dados existentes no documento WSDL, para classes java. É preciso ainda fornecer uma classe java que ofereça um método a cada operação definida.

A aplicação da abordagem *bottom-up* pressupõe a existência de um código em java, que deverá oferecer como um serviço web. Neste caso, é preciso criar um documento WSDL que mapeie o código em java. Novamente, é necessário um mapeamento entre os tipos java e o elemento <*types*> com base em XML no documento WSDL.

No caso dos solicitantes de serviços o processo a ser executado é análogo ao da abordagem *top-down*. Tem-se a definição de um serviço em WSDL. Para acessar esse serviço, deve-se criar as estruturas de dados apropriadas e interpretar os dados retomados, adequadamente. Os dois principais aspectos aqui são o mapeamento de tipos, no caso do provedor e a utilização da informação de vínculo fornecida.

Por exemplo, caso se tenha um vínculo SOAP no documento WSDL, deve-se criar um código de cliente java que envie o envelope SOAP correto ao serviço. A implementação Apache SOAP é um pacote que aceita isto.

Atualmente não existe nenhuma maneira padrão de fazer isso. Entretanto, têm sido empregados esforços para definir o mapeamento exato de um documento XML para um objeto java, e vice-versa. Na maioria das situações reais, este mapeamento é um processo muito direto. Além do mais, já existem ferramentas capazes de aceitar tanto a abordagem *top-down* quanto a *bottom-up*.

6.3.1 Mapeando termos de WSDL para termos do Java

Um documento WSDL descreve a interface de um serviço web, conforme visto anteriormente. No caso do uso de recursos de ferramentas que estão pautadas sobre o paradigma da orientação a objetos para auxiliar no desenvolvimento do serviço web, pode-se tentar mapear os termos que são usados na WSDL para o ambiente Java, por exemplo, conforme mostra o quadro 6.2. Esse mapeamento não é padronizado, sob nenhum aspecto, servindo apenas como uma diretriz.

Termo WSDL	Termo Java
<types>	Classe
<message>	parâmetros do método ou valor de retorno
<operation>	método
<portType>	Classe

Quadro 6.2 Mapeamento WSDL para Java

6.3.2 A WSDL para API java (WSDL4J)

A API WSDL4J é implementada através de um pacote denominado javax.wsdl e contém principalmente interfaces e algumas classes. Para usá-la, deve-se ter acesso não apenas a este pacote, como também a uma implementação das interfaces definidas, como por exemplo é oferecido no IBM Web Services Toolkit.

Apenas provedores de serviço precisam criar documentos WSDL. O solicitante de um serviço nunca criará. Logicamente é possível criar um documento WSDL manualmente, usando qualquer editor de texto ou editor de XML, mas o uso de ferramentas capazes de gerar o documento com base em código existente irá facilitar e agilizar o desenvolvimento do serviço web. O IBM Web Services Toolkit, categorizado como *freeware*, vem com uma ferramenta que gera documentos WSDL a partir de um código java existente. A plataforma GLUE possui uma ferramenta semelhante.

A criação do documento WSDL com o uso dessas ferramentas inclui algum tipo de introspecção das classes existentes do java, que são transformadas em definições de WSDL. Além disso, em alguns casos é necessário trabalhar com esquemas de XML já existentes, que descrevam as estruturas de dados do seu serviço. Essas podem ser copiadas para o documento WSDL ou importadas para ele, usando um elemento <*import*>.

6.3.2.1 Acesso ao documento WSDL

Os solicitantes de um serviço precisam acessar documentos WSDL para descobrir quais os serviços que eles oferecem e como eles podem ser chamados. Novamente o processo será agilizado pela utilização de ferramentas que analisem um documento WSDL existente e que crie algum código de cliente que possa integrar à aplicação do serviço web. O IBM web Services Toolkit e o GLUE, novamente, oferecem essa funcionalidade.

6.3.2.2 Vínculos Chamada dinâmica de serviço a partir da WSDL

A chamada dinâmica permite acessar documentos WSDL existentes, ou criar novos documentos a partir do Java, ou seja, não há a geração de código que possa chamar um serviço particular; em vez disso, tenta-se criar um cliente de serviço web que possa chamar qualquer serviço com base na sua definição de WSDL.

A chamada dinâmica de um serviço web é, na maioria das vezes, útil para testes ou cenários de avaliação onde se tenta descobrir como funciona um serviço particular.

O *Web Service Invocation Framework* (WSIF) permite a chamada dinâmica de um serviço web, com base unicamente na definição WSDL para esse serviço. Ele está disponível para *download* no site da IBM AlphaWorks.

O WSIF é baseado na API WSDL4l que foi apresentada anteriormente neste trabalho. Ele fornece classes adicionais que permitem analisar um documento WSDL e também usar seu vínculo definido e informações sobre a porta para criar uma chamada efetiva para o serviço.

6.3.3 GLUE

O GLUE é um ambiente completo para serviço web. O GLUE fornece suporte a todas as tecnologias de serviço web relevantes, como WSDL, UDDI e SOAP. Ele vem com um servidor web incorporado, um mecanismo SOAP, um servidor e cliente UDDI, e um console que permite a administração de serviços distribuídos e registrados. Todo o processamento em XML é feito usando Electric XML, que acompanha o GLUE, mas que também está disponível separadamente. A API aceita pelo Electric XML é semelhante à API JDOM. Serviços podem ser hospedados pelo ambiente do servidor HTTP contido por servidores de aplicação como BEA *webLogic*, ou por outros *containers servlet*, como Apache Tomcat.

O GLUE é implementado completamente em Java e aceita tanto clientes Java para chamar serviço web quanto objetos Java para serem expostos como serviço web

6.3.3.1 Publicando um serviço

A documentação do GLUE usa o termo "publicação" de um serviço. Neste caso, isso significa que um serviço está disponível em um servidor, e aceita solicitações do cliente. Isto não significa que o serviço seja publicado em um registro do UDDI. Em outros pacotes o termo usado é outro. Por exemplo, na implementação do Apache SOAP, um serviço é 'distribuído'.

Para publicar um novo serviço web, o GLUE precisa de um documento WSDL que o descreva. Pode-se criar um documento WSDL a partir de um código em Java já existente, usando uma ferramenta GLUE de linha de comandos, denominada java2wsdl.

6.3.3.2 Chamando um serviço

Para fazer uma chamada (ou seja, enviar uma solicitação SOAP) para um serviço, é necessária uma representação local desse serviço. Esse é um objeto *proxy* local para o qual se pode fazer chamadas, que a seguir encaminhará a solicitação ao serviço real. O uso de um serviço a partir de um programa do cliente requer duas etapas; a geração de uma interface Java local que fornece a interface de serviço em Java, usando a ferramenta WSD12JAVA. SUN (10)

7 REGISTRO DOS SERVIÇOS

Para que as empresas possam localizar os serviços oferecidos pelos serviços web, é necessário uma forma de torná–los públicos, ou seja, uma forma de registrar a existência dos mesmos no mundo web. Conforme foi apresentado no capítulo 5 deste trabalho os serviços web podem ser classificados em simples e complexos e, essa classificação leva a formas distintas de se realizar o processo de registro dos serviços web. Os serviços web simples se utilizam da tecnologia UDDI e os serviços web complexos, principalmente do eb–XML.

7.1 UDDI

A UDDI – *Universal Description, Discovery, and Integration* (Descrição, Descoberta e Integração Universais) consiste num padrão de registros (protocolos) para definições WSDL. Como parte do projeto UDDI, foi estabelecido um registro comercial global, a fim de permitir que os possíveis parceiros comerciais publiquem e descubram os serviços web. Ele fornece mecanismos para a busca de interfaces, utilizando seus atributos. A UDDI beneficia parceiros comerciais de todos os portes, pois possui uma especificação aberta de plataforma independente.

Mecanismo de registro e descoberta de serviços na web. Usado para armazenar e categorizar informações sobre serviços e para recuperar ponteiros para interfaces para serviços web, a UDDI é uma especificação global, seno que qualquer empresa espalhada pelo mundo é capaz de se registrar em um UDDI Business Registry.

Segundo definição de profissionais da ARIBA, empresa que juntamente com a IBM e outras pensou e sistematizou o UDDI o mesmo se constitui em um super–sistema de consulta de metadados que permite pesquisar fornecedores que correspondam aos requisitos desejados, fornecendo a indicação das redes comerciais a que estão ligados.

O UDDI são as páginas amarelas dos serviços web. Como as páginas amarelas tradicionais, pode–se procurar por uma companhia que ofereça os serviços necessários, ler

sobre o serviço oferecido e contatar alguém para obter mais informações. Um diretório UDDI é um arquivo XML que descreve o negócio e os serviços. O registro tem três partes: as "páginas brancas" descrevem a companhia: nome, endereço, contatos, entre outros, as "páginas amarelas" incluem as categorias, baseada em taxonomias padrões. As "páginas verdes" descrevem a interface para o serviço, em nível de detalhe suficiente para se escrever uma aplicação que use o serviço web. A forma como os serviços são definidos no documento UDDI é chamado *Type Model* ou *tModel*. Em muitos casos, o *tModel* contém o arquivo WSDL que descreve a interface SOAP do serviço web, mas o *tModel* é flexível o suficiente para descrever quase todo tipo de serviço.

Brancas, amarelas ou verdes, as páginas UDDI possuem um modelo de dados baseado em XML e o seu acesso exige a utilização do SOAP, tanto para as preencher como para pesquisar os serviços que contêm. Para provar que o UDDI funciona, a Ariba, a IBM e a Microsoft instalaram uma infra–estrutura de serviços de consulta distribuídos.

Cada uma destas empresas tem o seu site de serviços, que é sincronizado diariamente os sites das duas outras companhias. Adicionalmente, a IBM disponibilizou, em regime de código fonte aberto, classes Java para acesso ao UDDI.

Desde o seu lançamento mais de 120 empresas, entre as quais a HP, a Oracle e a Sun Microsystems, aderiram à iniciativa UDDI.

O diretório UDDI também inclui várias maneiras de procurar os serviços. Por exemplo, pode–se procurar por fornecedores de um serviço em uma locação geográfica específica ou por negócios de um tipo específico.

A analogia que se faz a este protocolo em face de uma arquitetura utilizando–se de J2EE ou mesmo CORBA são os protocolos JNDI Repos ou LDAP.

A especificação UDDI, conforme mostrado na figura 7.1 consiste de um esquema de 4 camadas dispostas hierarquicamente, as quais provêem um modelo base de informação para publicar, validar e invocar informações a respeito de um serviço web. São elas:

- Entidade de Negócio (*businessEntity*)– o elemento em nível mais elevado em um registro UDDI captura o conjunto inicial de informações requeridas por parceiros pesquisando para localizar informações a respeito de serviços de negócios, incluindo aí nome, o tipo de indústria, categoria de produto, sua localização geográfica, categorizações opcionais e informações de contato.

- Serviço de Negócio (*business services*) – esta estrutura agrupa uma série de serviços web relacionados de tal forma que eles possam ser relacionados a ambos processos de negócios quanto por categorias de serviços. Este agrupamento possibilita uma melhor otimização no tempo de pesquisa.

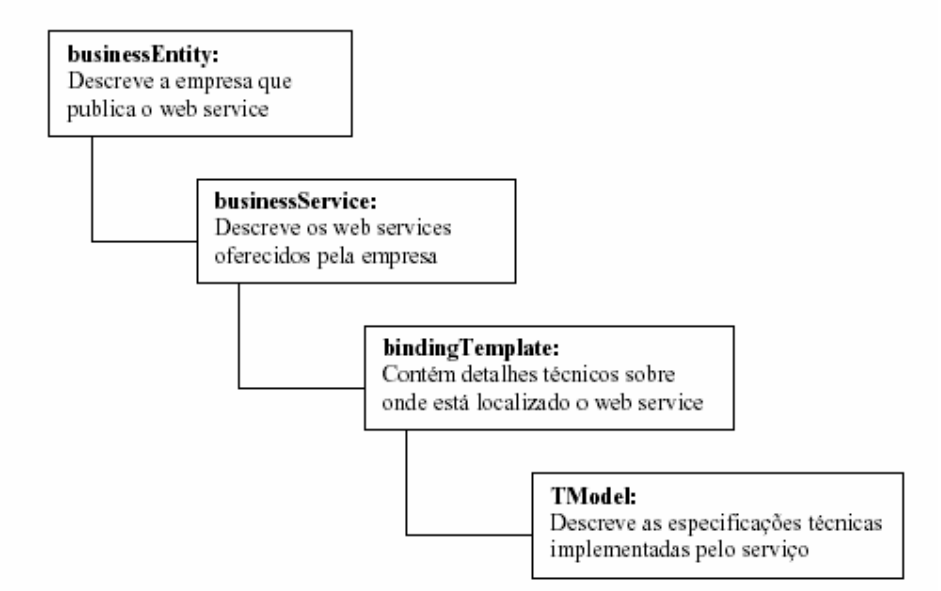

Figura 7.1 Estrutura UDDI

- Informações de troca (*bind Template*) – cada serviço web de negócio tem um ou mais elementos de descrições capturadas em XML, designados como exemplos de troca *(Binding Templates*). Este contêm informações que são relevantes para aplicações que são necessárias no momento de se invocar um serviço ou quando está ocorrendo uma troca com um destes. Estas informações referem–se, por exemplo, ao endereço URL do servidor.
- Informações de concordância (*TModel*) – são informações que possibilitam a um cliente determinar se um web Serviço específico está de acordo quanto à sua implementação. Referem–se a informações de ordem de compra do serviço, questões de segurança, que protocolos são apropriados, que tipo de resposta deverá ser aguardada após a ordem de compra, entre outras. UDDI (11)

7.2 eb-XML

Eletronic Business Extensible Markup Language, é uma variante XML desenvolvida para produzir um framework completo de transações B2B baseadas em XML.

Eb-XML construiu características de segurança para assegurar transações confidenciais. A fim de preparar o caminho para comunicações entre empresas, os padrões XML estão sendo definidos por diversas organizações. Por exemplo, a iniciativa *Electronic Business XML* (eb-XML) é uma colaboração entre a OASIS (*Organization for the Advancement of Structured Information Standards* — organização para o fomento de padrões estruturados de informação) e o UN/CEFACT (o órgão das Nações Unidas para a facilitação do comércio e dos negócios eletrônicos) para criar uma estrutura global em XML para o comércio eletrônico.

A eb-XML define um protocolo que lida com qualidades diferentes de serviço, desde mensagens de pouco peso (como no caso do SOAP) até mensagens de qualidade comercial (próprio para tratar de pedidos de compra entre empresas).

A eb-XML oferece uma maneira padrão de formatar as mensagens que voam para lá e para cá. É como se fosse um envelope. A tecnologia Java é uma das plataformas sobre a qual se lêem esses arquivos XML com invólucros de eb-XML e sobre a qual se processam as mensagens.

Quando duas empresas querem fazer negócios eletrônicos entre si, elas devem entrar em um acordo sobre como chamar os processos de negócio uma da outra e como vão trocar dados. Naturalmente devem também estar de acordo sobre protocolos e formatos de dados, assim como o conteúdo das mensagens que vão trocar entre si.

Uma outra questão é a publicação e descoberta dos serviços que as empresas tornam disponíveis a seus parceiros em potencial. E, além disso, fazer com que a interação entre processos de negócios mantenha a privacidade e a não interferência nos processos internos das empresas, quando não for desejado, e que contemple aspectos de segurança e tratamento de transações. Especialmente este último requisito dá margens para múltiplas ocorrências de erro que devem ser solucionadas como o tratamento de mensagens enviadas que não são respondidas, ou que são respondidas de forma não esperada.

A tecnologia dos serviços web é fracamente acoplada no sentido técnico, mas no sentido conceitual é o contrário. Qualquer coisa que afete a disponibilidade do serviço, a sua interface ou a semântica de uma mensagem vai implicar no fato de o provedor ter de notificar os seus clientes, o que não é uma tarefa fácil, uma vez que o provedor não tem o controle de quem são todos os seus clientes. Cabe ressaltar que não há necessidade de um contrato ou acordo entre partes para o uso dos serviços web.

Já o eb-XML é um padrão que recomenda um conjunto de especificações de negócio voltadas para a colaboração entre parceiros. Uma colaboração de negócio requer uma base sólida, abrangendo aspectos de acordos de negócio, assim como aspectos de infra-estrutura, de forma a prover um ambiente seguro e íntegro de troca de mensagens, com tratamento de transações e erros. De maneira análoga aos serviços web, eb-XML fornece interoperabilidade técnica, baseando–se em protocolos padrão. Entretanto, não se apóiam em WSDL. Ao invés de WSDL, utiliza CPAs (*Collaboration Protocol Agreements*) para declarar a ligação de suas interfaces de serviço com as suas especificações de colaboração de negócio, incluindo assim a semântica do processo de negócio na descrição de seus serviços.

Naturalmente, o processo de colaboração de negócios no eb-XML leva em conta condições de erro, uma vez que o serviço de mensagens do eb-XML prevê uma troca confiável e fornece mecanismos de recuperação de erros, ausentes na proposta de serviços web.

O acordo de troca de mensagens entre dois parceiros de negócios é descrito pelo CPA (*Collaboration Protocol Agreement*). O CPA pode ser considerado também como uma descrição de interfaces. Entretanto, se um dos parceiros altera a interface de um serviço de negócio descrito no CPA, ele coloca este CPA em um estado inválido, requerendo que um novo CPA seja elaborado entre os parceiros. Isso, porém não afeta tecnicamente a troca de mensagens. Desta forma, aquele que envia a mensagem pode ter certeza de que esta será entregue e que o recipiente está capacitado para lidar com o problema em potencial.

No mundo dos serviços web em uma integração que está situada nos limites internos da empresa, esse tipo de situação está sob o controle direto do pessoal de tecnologia. Eles estão a par das mudanças ocorridas nas interfaces dos serviços e podem adequar os sistemas envolvidos para lidarem com as mudanças. O mesmo já não ocorre em um contexto entre empresas.

Dessa forma, o eb-XML tem o seu ponto forte quando o processo de integração de negócios é entre empresas distintas, embora também seja adequado para integração entre divisões distintas de uma mesma empresa. Um outro aspecto é que, devido a modularidade do eb-XML, não há necessidade de uma empresa implementar todos os seus módulos. Pode–se implementar apenas o seu serviço de mensagens, por exemplo, para prover transporte seguro e confiável de dados.

Nos cenários de B2B os pontos fortes de eb-XML e serviços web podem ser combinados. eb-XML pode atuar no contexto de integração entre empresas, enquanto os serviços web, mais simples de implementar, podem ocupar seu lugar no contexto de integração dentro da própria empresa, em um ambiente mais controlado e, na implementação de serviços web simples.

8 SEGURANÇA NOS SERVIÇOS WEB

A segurança é um assunto importantíssimo: o que não é de se surpreender, quando se pensa em seu escopo e na sua complexidade - isso sem mencionar os custos. O mundo dos negócios está confiando cada vez mais na Internet como sendo o meio de transporte para suas comunicações e transações mais importantes, onde se deve evitar que o patrimônio das empresas seja apropriado indevidamente.

O W3C (*World Wide web Consortium*, http://www.w3.org/) considerou os riscos relacionados à segurança como estando em três grupos:

1. Bugs ou problemas na configuração localizada no servidor web, o que permite que usuários remotos não-autorizados:
    - Se apropriem de documentos confidenciais que não lhe dizem respeito.
    - Executem comandos na máquina host do servidor, o que os deixam livres para modificar o sistema.
    - Obtenham informações sobre a máquina host do servidor web, o que lhes permitirá invadir o sistema.
    - Efetuem ataques Denial ofService (DoS), deixando a máquina temporariamente indisponível.
2. Riscos na parte do navegador, incluindo:
    - Conteúdo ativo que trava o navegador, danifica o sistema do usuário, viola sua privacidade ou que apenas causa aborrecimentos.
    - O mau uso, intencionalmente ou não, de informações pessoais fornecidas pelo usuário final.

3. Interceptação de dados de rede enviados do navegador para o servidor, ou vice-versa, através de escuta clandestina de rede. Os usuários não-autorizados podem operar a partir de qualquer ponto no caminho entre o navegador e o servidor, incluindo:

    - A rede na parte do navegador da conexão.
    - A rede na parte do servidor da conexão (incluindo intranets).
    - O ISP (*Internet service provider*) do usuário final.
    - O ISP do servidor.
    - Qualquer provedor de acesso local do ISP.

Uma aplicação segura deve implementar alguns, se não a totalidade dos seguintes serviços de segurança:

- Identificação e autenticação

Quem é usuário/emissor/receptor, como saber se a identidade é legítima?

- Autorização

O usuário/emissor/receptor pode acessar esses dados/executar essa transação?

- Integridade

Os dados enviados são os mesmos que os recebidos?

- Privacidade

É possível ter a certeza de que ninguém leu os dados enviados?

- Não-repúdio

Como provar para terceiros que os dados realmente foram enviados?

Esses aspectos são de extrema importância e existem diversas técnicas e soluções com vista a garantir a implantação desses serviços. Os serviços web ainda não possuem a totalidade desses serviços implementados, mas existem diversas iniciativas e, muitos dessas implementações podem ser viabilizadas através dos recursos das linguagens de programação utilizadas para o desenvolvimento dos aplicativos que dão aporte aos serviços web.

O fato do SOAP ao usar a porta 80 do http, pode enviar requisições diretamente através do *firewall*, mostra que o modelo de serviços web alterou a situação referente à segurança, por um lado, os serviços web simplificaram as comunicações, mas ao fazê-lo, alteraram algumas das medidas de segurança já existentes.

A segurança pode ser tratada em diferentes camadas: desde a camada de rede de baixo nível até a de transporte, e finalmente a camada da aplicação propriamente dita.

8.1 Segurança na Camada de Transporte

A segurança pode ser obtida usando vários mecanismos de segurança incorporados a partir dos próprios protocolos de transporte na Internet. O HTTP (*Hypertext Transfer Protocol*) e o SMTP (*Simple Mail Transfer Protocol*) são exemplos disso, sendo que cada um é acompanhado por seu próprio conjunto de medidas de segurança. No entanto, há algumas restrições quanto ao uso dessa abordagem, especialmente quando ela se destina a proteger os serviços web.

8.1.1 Esquema de autenticação do HTTP

O esquema de autenticação do HTTP baseia-se no modelo em que o agente de usuário, ou navegador, deve autenticar a si mesmo com um nome de usuário e uma senha, para cada domínio, ou área, no servidor. Em termos de uma aplicação web padrão, o navegador receberia a mensagem de resposta 401 (não autorizado) enviada pelo servidor de origem para não permitir a autorização de um usuário, conforme definido pela especificação do HTTP, exibindo, assim, uma caixa de diálogo de não permissão. Esta caixa de diálogo solicita ao usuário informações sobre o nome de usuário e a senha. Quando os detalhes corretos são fornecidos, concede-se o acesso ao recurso.

Grande parte do processamento que usa um esquema de autenticação básico permanece oculto até mesmo para o desenvolvedor, pois a interação é manipulada de maneira transparente pelo HTTP. Da mesma maneira, os dados reunidos são limitados aos solicitados pelo HTTP, ou seja, o nome de usuário e a senha. Um outro método de autenticação, mais popular, é o oferecido pelos formulários de HTML.

8.1.2 Autenticação baseada em formulários de HTML

A abordagem dos formulários de HTML dá-se na camada acima do HTTP - HTML. Sendo assim, ele é mais flexível, o que significa não apenas que se tem um acesso mais programático à interação, mas também que fica mais fácil decidir quais dados podem ser reunidos, pois não há limitação aos campos do nome de usuário e da senha.

No caso dos serviços web, é preciso verificar se este método particular de autenticação é aceito. No caso da autenticação básica do HTTP/1.0, ele deve ser; uma alternativa para a caixa de diálogo padrão é um cliente SOAP que também poderia passar esta informação de volta para o servidor de origem, para que ela fosse autenticada. No entanto, é provável que os formulários de HTML não sejam aceitos por ferramentas de interface para serviços web.

Um outro aspecto a ser considerado, e que está associado tanto com o esquema de autenticação básico do HTTP/1.0 quanto com os formulários de HTML é a maneira como a combinação 'nome de usuário/senha' é transportada para o servidor de origem. No caso do esquema de autenticação básico do HTTP/1.0, os dados são codificados no formato base 64, um formato extremamente fácil de se decodificar. Com relação aos formulários de HTML, eles sequer são codificados, sendo, na verdade, enviados através do HTTP como texto puro.

Uma forma de se resolver esse problema consiste em combiná-los com o protocolo Secure Socket Layer.

8.2 Segurança da Camada de Aplicação/SOAP

Uma das maneiras mais diretas de se chegar à segurança no SOAP consiste em examinar os canais de transmissão usados para trocar as mensagens SOAP. O SOAP viaja através do HTTP, tanto dentro quanto fora das redes corporativas. Duas das maneiras mais populares de se proteger o HTTP são os usos do SSL ou das VPNs (*Virtual Private Networks*).

Usar o SSL é razoavelmente simples e não exige nenhuma alteração radical no código. O uso do SSL faz o servidor web trabalhar muito para criptografar e decriptografar as informações que são enviadas através do HTTP. Se a aplicação on-line for usada por milhares de usuários, o SSL deve ser usado cautelosamente para que não ocorra um forte impacto sobre os usuários que teriam, assim, uma resposta lenta para as páginas.

Para implementar o SSL, o administrador de sistema, pode registrar-se em um certificado SSL e configurar um servidor web, como o IIS ou o Apache, para comunicar através de um canal seguro.

No caso das VPNs, configura-se um túnel criptografado entre duas máquinas ou redes que permitam que a transmissão entre as duas extremidades da VPN seja segura. A VPN permite que os dados sejam transferidos através da Internet pública, de uma maneira segura que, para o usuário final, pareça que os dados têm origem em um membro da rede privada desse usuário. Uma vez que chega à outra extremidade, a comunicação pode ser vista na rede interna. A segurança está no canal usado para transmitir a mensagem.

As soluções de VPN se baseiam, geralmente, em IPSec, *Internet Protocol Security*, um *framework* de padrões abertos desenvolvidos pela IETF (*Internet Engineering Task Force*), para garantir a privacidade e a autenticação dos dados, além da autenticação do usuário, em redes públicas. Uma das principais vantagens da IPSec é que ela opera na camada de rede e permite que as aplicações operem de maneira transparente nas redes.

No caso de aplicações que operam com um número reduzido de instalações físicas e de clientes, uma opção VPN pode ser considerada mais apropriada. Caso o projeto precise de trocas com centenas de clientes distribuídos em várias redes públicas, é provável que o SSL seja a melhor opção.

Como acontece em muitos aspectos dos serviços web, a segurança é composta de padrões emergentes. Vêm sendo feitos muitos esforços em relação à padronização XML, no sentido de se obter uma representação comum das informações sobre segurança (como assinaturas, autenticação, autorização, gerenciamento de chave e outras).

A segurança em nível de aplicação requer mais trabalho do que a segurança de canal. O código deve ser alterado para fornecer segurança tanto da parte do remetente quanto da parte do receptor da aplicação. Pode-se oferecer um maior grau de proteção caso ele seja usado em conjunto com as opções de segurança do canal como SSL ou VPN. Geralmente, consegue-se a segurança em aplicações quando se aplica uma tecnologia de criptografia aos dados que são enviados e recebidos. Tanto o remetente quanto o receptor devem compartilhar o conhecimento sobre como a criptografia é feita, para poderem criptografar e decriptografar uma mensagem. Ao aplicar a criptografia estão sendo resolvidos os problemas referentes à confiabilidade no sentido de permitir uma troca de informações segura. Além disso, o código deve ser executado de acordo com o serviço ou o nome de usuário apropriado.

As cinco áreas apresentadas pela anotomia dos padrões SOAP a serem observadas para questões de segurança são:

- O formato da mensagem.
- A codificação.
- As convenções de RPC usadas.
- As vinculações de transporte -no caso, http.
- A natureza das mensagens com anexos.

O formato da mensagem está relacionado à maneira como a mensagem SOAP é empacotada antes de ser enviada através da rede. A mensagem contém aparte referente ao *Primary MIME* e aos anexos - a parte relativa ao *Primary MIME* compreende o SOAP Envelope, que, por sua vez, contém o cabeçalho e o corpo SOAP.

A codificação SOAP, conforme visto no capítulo 5, baseia-se no esquema XML: ela usa valores simples, construídos a partir de tipos oriundos do esquema XML, tipos de dados simples, enumerações e *arrays* de *bytes*. W3C (16)

8.2.1 Intermediários SOAP

O conceito de intermediários é um ponto importante do modelo SOAP para a troca de mensagens. Um intermediário é, basicamente, uma extremidade, um receptor de uma mensagem SOAP, que também pode enviar a mesma mensagem para outra extremidade. Os intermediários e o conceito de cadeia de mensagem introduzido por eles possibilitam o desenvolvimento de sofisticados sistemas baseados em SOAP. Um exemplo de intermediário pode ser um *Aggregation Service*, ou seja, um portal de informações que agrupa conteúdo de várias origens, algumas das quais podendo corresponder aos serviços web.

Os protocolos de transporte seguros, como SSL, TLS e IPSec, podem fornecer a integridade e privacidade da mensagem, durante a transmissão. Entretanto, devido ao fato das mensagens serem recebidas e processadas por intermediários, a comunicação fim a fim segura não é possível, se não existir uma associação confiável entre todos os intermediários, mesmo que os *links* de comunicação entre eles sejam confiáveis. A segurança fim a fim também fica comprometida, caso um dos *links* de comunicação não seja seguro.

O envolvimento dos intermediários e a necessidade da segurança fim a fim aumentam ainda mais a importância da camada da aplicação, no que se refere aos serviços de segurança. É onde entra o SOAP/segurança da camada de aplicação: para garantir a privacidade, a integridade e a validade dos dados, no contexto dos intermediários SOAP.

Um aspecto importante a se considerar é que com a segurança nas camadas de rede e de transporte, o elemento da mensagem SOAP não precisa ser alterado para se beneficiar do serviço de segurança. Porém, com a segurança da camada de aplicação/SOAP, é preciso modificar as mensagens SOAP propriamente ditas. As extremidades da aplicação de uma mensagem SOAP precisam estar aptas a discernir o tipo de segurança usada, além de como e quando aplicá-la. Definir e fornecer esses requisitos de mensagem constituem uma parte fundamental das várias iniciativas de segurança XML descritas a seguir.

8.3 Extensões de Segurança SOAP

Os módulos de extensão oferecidos pelo SOAP 1.1 oferecem um mecanismo pelo qual os recursos de segurança podem ser incluídos na mensagem SOAP propriamente dita. Obtém-se tal resultado por meio da combinação do elemento de cabeçalho SOAP com extensões de segurança.

Tendo sido publicado em 6 de fevereiro de 2001, o memorando do W3C *SOAP Security Extensions: Digital Signature* especifica as regras de sintaxe e de processamento

de uma entrada de cabeçalho SOAP para transportar informações de assinatura digital dentro de um envelope SOAP 1.1. Ele propõe que outros recursos de segurança, como a criptografia XML, sejam incorporados à medida que forem ficando disponíveis. W3C (16)

8.4 Assinatura XML

Do ponto de vista do W3C, foi realizado um maior progresso para as assinaturas XML do que para extensões de assinatura, o que é confirmado pelo fato de que um work group do W3C formou-se em torno delas.

A missão de um WG de assinatura XML consiste em desenvolver uma sintaxe compatível com XML, e que seja usada para representar a assinatura dos recursos web e dos trechos de mensagens referentes ao protocolo e aos procedimentos para o cálculo e a verificação dessas assinaturas.

As assinaturas XML aplicadas aos dados dentro do mesmo documento XML que a assinatura são chamadas de assinaturas de envelopamento ou envelopadas; já as assinaturas aplicadas aos dados externos ao elemento da assinatura são chamadas de assinaturas destacadas.

O número de possibilidades, como a ordem segundo a qual a criptografia, a assinatura, a modificação e, talvez, mais assinaturas, possam ocorrer, é enorme. Partindo-se da idéia dos intermediários SOAP é possível conceber aplicações requeridas, em uma situação onde outro intermediário tenha desistido, podendo, hipoteticamente, decriptografar dados, recriptografar dados decriptografados pela extremidade atual, ou ambas as ações. Isso precisa ser feito de maneira que não impeça que outros intermediários, ou o próprio receptor, decriptografem os dados. W3C (16) XML (15)

8.5 Criptografia XML

A idéia da criptografia XML consiste no seguinte:

- Um processo para criptografia/decriptografia do conteúdo digital incluindo fragmentos/documentos XML
- Uma sintaxe XML usada para representar o conteúdo criptografado e informações que permitam que o receptor o decriptografe.

A partir desse WG, encontram-se acessíveis os seguintes documentos:

- XML Encryption Requirements Working Draft, em http://www.w3.org/TR/xml-encryption-req/
- XML Encryption Syntax andProcessing WorkingDraft, em http://www.w3.org/TR/xmlenc-core/

- Decryption Transformfor XML Signature Working Draft, em http://www.w3.org/TR/xmlenc-decrypt/

8.6 XKMS

A XKMS (*XML Key Management Specification*) foi submetida ao W3C em março de 2001. Ela especifica protocolos para distribuir e registrar chaves públicas, adequadas para usar em conjunto com o padrão proposto para a assinatura XML e um padrão associado para a criptografia XML. Também define protocolos para a resolução/recuperação de chaves públicas, e associação e recuperação de atributos na forma de asserções confiáveis com chaves públicas.

O objetivo consiste em fornecer uma interface de nível mais elevado, e mais flexível, para serviços confiáveis do que os atualmente fornecidos pelas infra-estruturas de chave pública de primeira geração, conforme já descrito anteriormente.

A XKMS inclui, essencialmente, três serviços e especificações: *Key Information Service Specification* X-KISS, *Key Registration Service* X-KRSS e *Trust Assertion* Service X-TASS.

A X-KISS fornece um serviço de resolução/recuperação e verificação de chave pública. Serviços X-KISS de baixo nível permitem que aplicações recuperem o certificado de um usuário. Já serviços de alto nível fornecem uma interface para um intervalo de informações vinculadas ao dono de uma chave pública identificada sendo que as informações fornecidas com suporte a detalhes confiáveis.

A X-KRSS define um protocolo para o registro das informações de chave pública. Um usuário que registra um serviço talvez precise que informações adicionais sejam ligadas à chave pública; podem ser um nome, um identificador ou atributos estendidos definidos pela implementação.

A especificação X-TASS define uma arquitetura e um protocolo para as instruções de recuperação, conhecidas como *trust assertions* (ou asserções confiáveis), que são ligadas à chave pública.

Com a SAML, a XACML/XACL, e outras iniciativas, a XKMS é uma peça importante no amplo quebra-cabeça que forma a segurança, aplicada a documentos XML. Seu efeito imediato é a enorme simplificação do gerenciamento de chaves de autenticação e assinatura; isso é feito por meio da separação da função do processamento de certificado digital, desconsiderando a verificação de status, a localização e a validação do caminho de certificação a partir da aplicação envolvida, por exemplo, delegando o gerenciamento de chave para um serviço web na Internet. W3C (16) XML (15)

8.7 Outras Iniciativas de Segurança para XML

8.7.1 SAML

O trabalho posto em prática pelo SSTC (*Security Services Technical Committee*) fundamentado na XML OASIS reúne duas iniciativas, anteriormente concorrentes, a S2ML e a AuthXML. A especificação resultante de tal processo, a SAML (*Security Assertion Markup Language*), permite que as empresas troquem, com segurança, informações sobre autenticação, autorização e perfil com seus parceiros.

8.7.2 XACML/XACL

A EACML *(Extensible Access Control Markup Language*) é uma especificação que permite que as empresas restrinjam o acesso a serviços a usuários autenticados e autorizados, o que se assemelha muito à SAML, embora esteja muito mais relacionada a um modelo de segurança orientado a assunto-privilégio-objeto, no contexto de um determinado documento XML. Ao escrever regras em XACML/XACL, a pessoa que cria as políticas de acesso pode definir quais pessoas exercem que privilégios de acesso, para um determinado documento XML, o que é de grande importância nas situações citadas anteriormente.

8.7.3 P3P (platform for privacy preferences project)

A P3P, plataforma para a criação de preferências privadas, permite que os sites expressem suas práticas de privacidade em um formato padrão, que pode ser automaticamente recuperado e interpretado com facilidade por agentes de usuário. Os agentes de usuário P3P permitirão que os usuários sejam informados sobre as práticas do site (nos formatos legível pela máquina, e por humanos), para automatizar a tomada de decisões com base nessas práticas, quando for apropriado. Assim, os usuários não precisam ler as políticas de privacidade de todos os sites que visitarem.

8.8 APIs, Kits de Ferramentas e SDKs de Fegurança do Java

Os códigos, kits de ferramentas e SDKs para Java disponíveis para trabalhar com as tecnologias dos serviços web podem ser divididas, de forma geral, em dois tipos: as usadas dentro do âmbito dos serviços web e as usadas na comunicação de um serviço web.

Algumas das APIs de segurança java da Sun se enquadram no primeiro grupo são usadas nos kits de ferramentas da segurança XML mencionados a seguir.

O *Standard Edition Development Kit*, na versão 1.3 apresenta como pacotes opcionais para *download*, disponíveis as seguintes APIs de segurança:

- O JASS (*Java Authentication and Authorization Service*) é um conjunto de pacotes que permitem que os serviços autentiquem e imponham controles de

acesso para os usuários. Ele implementa uma versão do java padrão *Pluggable Authentication Module framework*, além de fornecer suporte para controles de acesso baseados no usuário, no grupo ou na função.

- JCE (*Java Cryptography Extension*) é um conjunto de pacotes que fornecem um *framework* e algumas implementações para criptografia, criação e acordo de chave, e algoritmos MAC (*Message Authentication Code*). O suporte para criptografia inclui cifrados simétricos, assimétricos, de bloco e de fluxo. O software também aceita fluxos seguros e objetos selados.
- JSSE (*Java Secure Socket Extension*) é um conjunto de pacotes que propiciam comunicações seguras de Internet, além de implementar uma versão para Java de protocolos SSL (*Secure Sockets Layer*) e TLS (*Transport Layer Security*) e apresentar uma funcionalidade referente à criptografia de dados, à autenticação de servidor, à integridade de mensagem e à autenticação do cliente. Ao usar a ISSE, os desenvolvedores proporcionam a passagem segura dos dados entre um cliente e um servidor que estejam executando qualquer protocolo de aplicação (como o HTTP, o Telnet, o NNTP e o FTP) no TCP/IP. Maiores informações podem ser encontradas em: http://java.sun.com/products/jsse. Sun (10)

Cada uma dessas APIs é integrada ao J2SE, versão 1.4.

9 ORIENTAÇÃO A OBJETOS E ARQUITETURA ORIENTADA A SERVIÇOS

Aplicadas as pesquisas para averiguar o estado da arte das tecnologias envolvidas no escopo desse trabalho é possível realizar os estudos, comparações e constatações teóricas aos quais esse trabalho se propôs.

Nos anos 90 a programação orientada a objetos permitiu encapsular dados e funções e dar vida a estas entidades. Surgem dai os ícones e janelas na tela do microcomputador como formas da comunicação humana com estes objetos virtuais. Cria-se, no ambiente de trabalho virtual, analogias com o mundo real dando o impulso que a orientação a objetos precisava para se firmar como um tecnologia presente em todas as novas aplicações computacionais.

9.1 Da Mobilidade dos Projetos a Reusabilidade de Aplicações

Traduzir o mundo orientado a objetos exige uma anotação poderosa, preferencialmente gráfica, que permita representar os detalhes dos sistemas reais e ao mesmo tempo ser uma transcrição para uma linguagem de programação executável. A computação viu no final dos anos 90 surgir a UML, como anotação padrão para descrição de sistemas OO, e que permite transpor, com alguma facilidade, as estrutura e relações presentes no mundo real para os sistemas computacionais. Apoiada em ferramentas CASE, a UML está hoje presente em todos os projetos como uma planta baixa do software, trazendo para as fases iniciais do desenvolvimento algumas das decisões mais importantes de projeto. Essa padronização já presente desde o nível de projeto traz inúmeras vantagens para o todo de qualquer processo, entre elas: reusabilidade e mobilidade.

Mobilidade é a palavra de ordem da computação atual. Incentivada pela Internet, os objetos são convidados a viajar pela rede entre computadores, para servir com suas funcionalidades usuários remotos. Garantir esta mobilidade exige além de um projeto

cuidadoso na UML a portabilidade nas linguagens de programação, que possa servir e receber estes objetos. Assim se constituiu o conceito de objeto distribuído, e com ele surgiu uma sofisticada tecnologia para o seu desenvolvimento e a sua implementação. Disputaram e, em muitas aplicações ainda disputam a atenção dos desenvolvedores à tecnologia CORBA, DCOM, RMI e os componentes EJB.

A OO coloca sua marca nas propostas de interoperabilidade oferecidas pelas tecnologias citadas, mas ainda falta algo que contribua de fato para a existência real de tal interoperabilidade. Surgem, um pouco mais recentemente, os serviços web, como uma alternativa ao acesso remoto de serviços, integrando os objetos com a Internet, unindo assim dois dos mais novos paradigmas do mundo do software: OO e SOA.

9.2 Evolução dos Middlewares para Serviços Web

É inegável que os serviços web trazem em seu bojo uma grande transformação na Internet, com eles torna-se possível que as transações entre aplicações aconteçam com grande velocidade e facilidade, isso porque através deles se tem acesso a processos remotos utilizando um padrão aberto (XML). Contudo é importante se avaliar as contribuição de paradigmas e *middlewares* até então existentes, pois não será possível em hipótese alguma ignorar o que existe funcionando, pelo contrário é preciso a integração do novo com o que já esta em uso.

Essa relação de comunicação entre os *middlewares* que já existiam e a nova tecnologia dos serviços web é possível se avaliar que existe um processo de integração entre esses dois paradigmas.

O SOAP, que é um dos principais protocolos que dá sustentabilidade ao uso e aproveitamento dos recursos dos serviços web, efetua o acesso através da invocação de um serviço, ou seja, de um método e o WSDL, protocolo utilizado para descrever o serviço, possui na sua essência a descrição de como se deve definir os métodos de acesso a esses serviços e, logicamente essa forma de operação é uma herança precisa do mundo OO.

9.3 Características da Arquitetura Orientada a Serviços e a Garantia de Interoperabilidade

A arquitetura orientada a serviço se caracteriza principalmente pelo fato de que o usuário de um serviço web necessita apenas conhecer a descrição do serviço em si e mais nada, ele está na busca de um serviço e da forma de invocá-lo. Novamente aqui, pode-se constatar a presença clara de conceitos voltados a OO, pois após definições e estruturas o que se faz com objetos simplesmente é invocá-los.

Além disso, os serviços web por si só não se apresentam como soluções completas seja para EIA, e-*commerce*, CRM (*Customer Relationship Mangement*) ou ainda para qualquer outra aplicabilidade, pois antes de se oferecer os métodos como serviços web são necessários que eles existam na sua estrutura enquanto aplicativo e, isso, comprovadamente, na maioria das grandes empresas é realizado de forma orientada a objetos, desde a parte que envolve a análise dos requisitos até o desenvolvimento em si da aplicação, donde se pode ainda perceber de forma clara a presença cada vez maior de padrões para a realização dessas atividades.

A correta elaboração de um sistema, que logicamente envolve fases de modelagem inclusive na sua grande maioria com uso de UML garante o oferecimento de aplicações íntegras que podem disponibilizar seus métodos para serem oferecidos em forma de serviços web, garantindo dessa forma a interoperabilidade e a reusabilidade entre aplicações distintas.

Outro ponto a considerar é a elaboração dos arquivos de descrição dos serviços (WSDL), de acesso a esses serviços (SOAP) e ainda de registro dos mesmos. Inicialmente, como todo os processos de criação de software, esses arquivos foram criados a partir de editores de textos comuns. No entanto, hoje muitas pessoas se utilizam de *toolkits* para criar, ler e analisar tais documentos e mensagens. Esses *toolkits* na sua totalidade, pelo menos no que diz respeito aos analisados para a elaboração desse trabalho, foram elaborados a partir de linguagens e ferramentas voltadas à orientação a objetos, devido à facilidade de uso e mapeamento dos recursos necessários. Isso garante não só uma forma diferenciada de trabalhar com a geração de serviços web, mas uma forma de mapeamento direto e dinâmico entre os aplicativos em si e os serviços oferecidos, pois não há mais a necessidade de interferência dos desenvolvedores para a geração dos serviços, sendo desnecessário o conhecimento mais profundo inclusive das estruturas SOAP, WSDL e UDDI, pois as mesmas já se encontram mapeadas em forma de classes na maioria das ferramentas utilizadas para desenvolvimento. Essa é uma prova incontestável da integração que existe entre esses dois paradigmas tão importantes para o futuro da Internet.

Essa flexibilidade permitida pelas ferramentas também contribui para a solução de um outro problema apontado anteriormente: a existência de sistemas desenvolvidos em períodos diferentes, e que não podem ser substituídos, sendo que a partir das possibilidades de mapeamento esses sistemas também poderão oferecer os seus serviços de modo dinâmico na web. No entanto esses procedimentos são aplicados na sua grande maioria para sistemas que trabalham sobre o paradigma de orientação a objetos, onde se tornará mais fácil o mapeamento dos métodos das aplicações para serviços web.

Novamente pode-se constatar a integração e a força que essas novas tecnologias representam nesse novo mundo web.

9.4 Aplicabilidade SOA

As ferramentas de EAI (*Enterprise Application Interchange*) podem ser consideradas um outro exemplo positivo da integração dos paradigmas OO e SOA, pois muitas dessas ferramentas são desenvolvidas sobre o paradigma OO, com o uso de recursos de modelagem que permitem visualizar graficamente todo processo, bem como possibilita que as alterações sejam realizadas na própria interface gráfica. O uso de ferramentas de EAI estão aos poucos sendo direcionado para orquestrar os serviços web, o que poderá se constituir num grande passo para uma integração controlada podendo garantir às empresas o fluxo de uma operação B2B (*Business to Business*) mapeado e controlado, utilizando-se conforme já afirmado dos recursos da arquitetura de serviços web da Internet e da OO.

A orquestração de serviços web permitirá ainda a gerência dos processos entre as aplicações de longo curso, que pela imprevisibilidade dos seus tempos de resposta devem obrigatoriamente trabalhar desconectados. Para que se possa ter transações com estes elementos é necessário desenvolver os tratamentos de exceções e processos complementares, que possuem fácil implementação nas linguagens orientadas a objetos.

Nas interações com processos remotos onde se encontram transações complexas que necessitem alta disponibilidade e escalabilidade, é necessário o uso da orquestração de serviços web, visando assim à construção de sistemas dinâmicos, transacionais, com tolerância à falha com pleno acesso e integração com as aplicações e sistemas pré-existentes, o que certamente levará ao desenvolvimento de uma ferramenta criada para este fim. Aqui apenas a arquitetura SOA mais uma vez não resolveria o problema sozinha. Muitas das linguagens de programação orientada a objetos possuem o recurso do uso de *threads* que pode auxiliar em muito na questão de controle de fluxos, dentre inúmeras outras API´s que poderão ser utilizada para a resolução desses problemas.

9.5. Garantia de Reusabilidade com o Uso do SOAP Enquanto Framework

No decorrer das pesquisas realizadas foi possível encontrar em vários locais a definição de que o SOAP é um *framework* XML. Neste sentido cabe analisar o que vem a ser um *framework*.

Com o crescente desenvolvimento das áreas de hardware, software e das áreas de telecomunicações, uma nova abordagem de desenvolvimento de sistemas faz-se necessária para que se possa ter produtos de softwares mais confiáveis, portáveis e,

principalmente, para que se possa construir softwares de forma rápida e competitiva para um mercado hoje muito mais veloz e exigente.

Um dos principais objetivos da Engenharia de Software é a reutilização de sistemas já existentes. A orientação a objetos fornece facilidades para que classes possam ser reutilizadas, bem como métodos por meio de seus mecanismos de herança e polimorfismo.

Com a necessidade de um desenvolvimento rápido e confiável, surgiu uma nova tecnologia em Engenharia de Software que permite que uma família de produtos de softwares possa ser gerada a partir de uma única matriz. Essa nova tecnologia chama-se frameworks. CRE (03)

Algumas definições de frameworks:

- Um *framework* é um conjunto de objetos que colaboram com o objetivo de atender a um conjunto de responsabilidades para uma aplicação específica ou um domínio de aplicação.
- Fr*amework* é uma arquitetura desenvolvida com o objetivo de atingir a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização.
- Um *framework* é um conjunto de classes que constitui um design abstrato para solução de uma família de problemas.
- Outra definição de *framework* é um software parcialmente completo (subsistema) projetado para ser instanciado. Isto define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. Também se definem os lugares ou pontos nos quais uma adaptação do código para um funcionamento específico de certos módulos deve ser feita.

Portanto, um *framework* é uma aplicação semi-acabada que, muitas vezes, pode ser um sistema inteiro ou, às vezes, um subsistema. Ele procura descrever os objetos e suas interfaces presentes no design, fornecendo informações sobre seus fluxos de controle entre as diversas classes, e mapeia, também, as responsabilidades dos objetos. UNI (14)

Frameworks também promovem reutilização de implementações; porém, isto é menos importante do que as suas interfaces internas e a maneira pela qual elas se comunicam. Este é um reuso de alto nível de *design*, e *frameworks* são uma boa abordagem para esta proposta de reutilização. Tipicamente um *framework* é implementado utilizando-se alguma linguagem de programação orientada a objetos tipo Java, C++, Smalltalk, Eiffel, dentre outras.

Uma das características dos *frameworks* é a inversão do controle. Tradicionalmente, um desenvolvedor reutiliza componentes de uma biblioteca em que o programa principal chama os componentes quando necessário. O desenvolvedor decide quando e como o componente será chamado e sua interação com os demais. Em um *framework*, o programa principal é reutilizado e o desenvolvedor decide o que será plugado dentro dele, determinando a estrutura geral e o fluxo de controle dos programas. UNB (13)

Os benefícios primários que se pode identificar no uso de *frameworks*, segundo Fayad, Jacobson, Kristensen, Nowack, Froehlich, Hoover, Liu e Sorenson são:

- Modularidade: *frameworks* aumentam a modularidade, pois encapsulam detalhes de implementação sob interfaces bem definidas e estáveis. Esta modularidade auxilia na qualidade do *software*, localizando os lugares de impactos no *design* e nas trocas de implementações, reduzindo o esforço necessário para entender o *design* e para realizar futuras manutenções;
- Reutilização: interfaces estáveis presentes nos *frameworks* aumentam o potencial de reutilização pela definição de componentes abstratos que podem ser redefinidos para criarem novas aplicações. O aproveitamento dos componentes já definido aumenta também a produtividade dos programadores, aumentando, por sua vez, a qualidade, desempenho, confiabilidade e interoperabilidade do software;
- Extensibilidade: um *framework* aumenta a extensibilidade na medida em que oferece métodos *hook* explícitos. Estes permitem que aplicações possam ser estendidas usando-se as interfaces estáveis já presentes. (03)

Diante das definições e vantagens apresentadas sobre os *frameworks* e, avaliando-se a estrutura do SOAP, pode-se afirmar que muitas das definições acima realmente o caracterizam como um *framework*, pois ele é aberto, destacando-se características como reusabilidade e modularidade. Novamente aqui se nota um claro relacionamento entre OO e SOA.

9.6 Ferramentas de Desenvolvimento para SOA e Paradigma OO

O movimento de serviços web tem sido focado em promover a colaboração B2B via processos de negócios integrados. Mas o real benefício dos serviços web pode ser liberado na tecnologia de BI *(Business Intelligence*) e não no processamento de transação.

Companhias estratégicas que desenvolvem aplicativos de negócios assim como Oracle, SAP, PeopleSoft, Siebel Systems entre outras, estão construindo *frameworks* de

serviços web nos seus produtos podendo com isto estimular a demanda de BI em rede, através da mudança de cultura do relacionamento corporativo no comércio colaborativo.

Microsoft está promovendo .NET como iniciativa de serviço web, que requer o uso de software baseado em Windows. Sun Microsystems, de outro lado, está liderando uma estratégia de serviços web orientada em Java, suportado por mais 30 outros vendedores, o Sun One.

Forte desenvolveu um modulo de desenho de serviços web para ajudar no desenvolvimento de EJB´s e o registro de serviços web baseados em XML.

Oracle e seu produto Oracle9i Application Server release 2 tem suporte a serviços web padrões (SOAP, WSDL, UDDI) e também tem compatibilidade com Microsoft.NET.

Business Objects está escrevendo um SDK (*Software Developer Kit*) para serviços web baseado em Microsoft.NET, que deverá permitir que clientes construam o que a companhia chama de "2ª Geração de *Extranets* de BI".

Nessas iniciativas fica claro o uso das tecnologias dos serviços web e a base oferecida pela orientação a objetos na grande maioria das soluções apresentadas, pois as analisando a fundo, vê-se claramente que a essência do desenvolvimento se volta para linguagens orientadas a objetos ou projetos pautados em cima de modelagens muito bem elaboradas.

Na realidade o que se pode perceber ao longo do trabalho é que a grande maioria dos *toolkits* e ferramentas completas, envolvendo diversos fornecedores estão na sua grande maioria adotando o Java como uma ferramenta essencial de desenvolvimento. A própria Microsoft esta utilizando a riqueza da estrutura de classes oferecida pelo Java através da adaptação realizada sobre a linguagem no lançamento do C# (C Sharp).

9.6.1 Integração Java e XML

Segundo informações da Gardner cerca de 62% dos desenvolvedores que atuam nas empresas brasileiras usam a tecnologia Java. Em outra previsão o instituto afirma que 50.000.000 de celulares 3G com Java deverão estar no mercado até o final de 2003.

Isso comprova que a tecnologia Java esta se consagrando como uma das melhores ferramentas de desenvolvimento multi-plataforma que existe.

A linguagem Java é hoje, sem sombra de dúvidas uma das linguagens de programação mais completa e modular que existe. Suas diversas API´s contemplam praticamente todas as necessidades envolvidas com tecnologia de informação e a sua

estruturação em classes e pacotes permite uma evolução constante, possibilitando uma rápida adaptação a novas necessidades no mercado de aplicações.

Devido a enorme gama de recursos que oferece através dos inúmeros pacotes e classes o Java possui sustentabilidade para trabalhar com sistemas distribuídos e multi-plataformas de forma simples e direta, logicamente se utilizando para isso das vantagens da orientação a objetos, dentre as quais, extremamente importantes para as aplicações aqui citadas estão à herança e o polimorfismo.

Para solucionar um outro problema que seria a interoperabilidade, como foi visto no decorrer de todo o trabalho, tem-se a XML que é sem dúvida o protocolo universal de troca de dados pela Internet.

As tecnologias Java e XML possibilitam a entrega de serviços de web à empresa por ajudarem a simplificar o desenvolvimento de aplicativos focalizados em redes e baseados em padrões abertos para o setor.

- APIs em Java para XML permitem que os desenvolvedores alavanquem a XML facilmente para aplicativos comerciais eletrônicos.
- A construção de aplicativos com o uso de tecnologias baseadas em padrões abertos reduz os riscos de desenvolvimento e os custos de implantação, e ajuda a assegurar uma base sólida para o crescimento futuro.

A implantação de aplicativos baseados na tecnologia Java, que aproveitam as sinergias do código reutilizável e portátil da plataforma Java e dos dados reutilizáveis e portáteis da XML, e da interoperabilidade dos serviços web ajudam a simplificar a atualização e a manutenção dos sistemas de empresas. Assim sendo pode-se afirmar claramente que o uso conjunto das tecnologias OO e SOA garantem a interoperabilidade entre aplicações distintas.

Analisado-se a arquitetura orientada a serviços sob o prisma da OO pode-se dizer que a UDDI define uma classe com métodos abstratos, dos quais um desses métodos quando concretizado implemente a estrutura da WSDL, que por sua vez, nos procedimentos descritos definem novas classes abstratas, que são concretizadas através da definição dos arquivos SOAP.

Essa comparação logicamente é muito simplória, mas pretende mostrar que os conceitos envolvidos nos serviços web advém do mundo OO. Além disso como foi visto no decorrer do trabalho o uso de ferramentas e linguagens OO com as tecnologias da arquitetura orientada a serviço, garantem a interoperabilidade e a reusabilidade entre

aplicações diversas, permitindo inclusive a utilização de aplicações web já existentes para disponibilizar os serviços web.

9.7 O futuro dos Serviços Web

A previsão do META Group para serviços web é que os padrões inicialmente desenvolvidos para propiciar a integração e interoperabilidade (XML, WDSL, UDDI) chegarão ao biênio 2005/2006 com o status de arquitetura orientada a novos serviços.

Segundo o META Group até 2004, novas aplicações corporativas surgirão de componentes baseados ou integrados às tecnologias Java 2 Enterprise Edition (J2EE) e a Microsoft.Net, e que os fornecedores destas soluções devem se diferenciar com a oferta de valor agregado, como serviços web e de desenvolvimento de produtos e/ou ferramentas específicas.

Pesquisas do Forrester Research Inc. mostram que tecnologias tradicionais de EAI deverão dominar até 2004, enquanto serviços web melhoram o seu desempenho. Por volta de 2006, companhias devem adotar serviços web básicos, e ampliar o uso da tecnologia para transações mais complexas. Analistas da Forrester aconselham para iniciar o trabalho em serviços web básicos com parceiros e internamente, monitorando a evolução de padrões de serviços web para identificar capacidades mais promissoras para mais tarde adotar.

Os serviços web devem simplificar o processo de integração de aplicativos e processos comerciais nos próximos anos. O início dessa caminhada se deu com a implantação de diversos serviços web simples que são utilizados diariamente por inúmeros internautas sem que eles se dêem conta disso, como por exemplo os recursos disponibilizados pelo portal de busca do Google. A evolução natural desse caminho deverá ser o aprimoramento das tecnologias que envolvem os serviços web complexos, principalmente em questões de segurança e controle de fluxo. Com certeza nesses processos estarão novamente presentes os conceitos de orientação a objetos, pois na nossa realidade tecnológica esse é o paradigma presente na maioria das soluções computacionais.

9.8 Serviços Web – Dinamicidade, Interoperabilidade e Reusabilidade com SOA e OO

Com a emergência da Internet de próxima geração, a web deixará de ser composta por sites independentes em que o internauta navega à procura dos produtos pretendidos, e passará a ser composta por numerosos serviços oferecidos às empresas

através da Internet. Cada um destes serviços será uma aplicação modular, em conformidade com um formato técnico normalizado, que desempenhará uma tarefa precisa e que poderá ser encontrado de forma dinâmica num site de serviços.

As interfaces destes serviços assumirão a forma de objetos. Apresentarão os serviços disponíveis, independentemente da sua concretização, e permitirão o acesso a tais serviços a partir de qualquer ponto de acesso à Internet. Estes últimos irão constituir-se em recursos reutilizáveis e acessíveis através de uma URI ou outras estruturas que venham a ser definidas, o que evitará descrever os mesmos serviços em cada empresa.

Os serviços web são componentes de softwares que oferecem as funcionalidades acima descritas. A grande vantagem é que esses componentes podem ser acessados por diferentes sistemas, através de padrões da internet.

A diferença básica entre os serviços web que usam os recursos acima descritos e outras iniciativas como CORBA, DCOM e RMI, está na interoperabilidade das plataformas e aplicações, pois apesar dessas iniciativas prometerem essa interoperabilidade a mesma não ocorreu de forma independente, pois o CORBA, como exemplo, requer a instalação do ORB nas estações e o RMI, apesar de possibilitar interoperabilidade entre plataformas distintas prende as aplicações a linguagem Java.

Na evolução histórica do software pode-se constatar que o grande avanço para se chegar aos middlwares até então disponíveis passaram pelas técnicas estruturadas e, por último obtiveram um grande avanço utilizando-se para isso do paradigma de orientação a objetos. Tanto CORBA, DCOM, quanto RMI são baseadas em arquiteturas e recursos advindos do uso de objetos como fonte para as facilidades apresentadas.

9.8.1 Contribuições do mundo OO

Os serviços web não renegam essa herança do mundo da orientação a objeto, pelo contrário as aproveitam de modo especial e criam um novo paradigma que trabalhara de modo conjunto à orientação a objetos: a arquitetura orientada a serviços.

Os serviços web permitem que as aplicações das empresas façam uso de todos os recursos oferecidos pelo mundo da orientação a objeto para mostrarem as suas funcionalidades. Isso pode ser facilmente comprovado ao se analisar as linguagens e ferramentas de programação disponíveis para o desenvolvimento dos aplicativos que servirão como base para a interconexão dos serviços Web. A Sun possui toda a sua estrutura 100% voltado ao mundo dos objeto. A linguagem Java, carro chefe de todas as

API's e ferramentas oferecidas pela SUN é totalmente orientada a objeto. Outras empresas como IBM e Oracle estão incluindo em suas ferramentas o Java como forma de interação e muitos dos conceitos de orientação a objetos já estão presentes em várias de suas ferramentas e são até mesmo suportadas pelos bancos de dados. A Microsoft já vem ao longo dos anos voltando as atualizações de suas linguagens de programação como o Visual Basic para o uso dos benefícios de objetos e, o lançamento do C# (C Sharp), na realidade uma linguagem Java reescrita não deixa dúvidas sobre suas intenções de aproveitar as vantagens da orientação a objetos.

Assim sendo a funcionalidade das aplicações que se constituíram em serviços web está na sua grande maioria sendo possibilitada pelo uso de ferramentas voltadas ao mundo dos objetos. Entretanto para que a interoperabilidade entre essas aplicações seja possível é necessário o uso de novos recursos, os recursos orientados a serviços. Ou seja, as aplicações não precisam se preocupar com detalhes das implementações das funções de outras aplicações das quais apenas querem utilizar o seu serviço. Precisam sim, preocupar-se com a forma correta e ágil de localizar esses serviços, definir a forma de como acessa-los e o faze-lo. Isso constitui a essência dos serviços web.

Diante desse quadro pode-se observar que existe uma integração entre o paradigma de orientação a objetos e a arquitetura orientada a serviços, ou seja a orientação a objetos esta presente até o momento em que a função é traduzida do seu sistema para a linguagem universal dos serviços web, a XML, processo conhecido como *marshal.* No momento em que isso ocorre a localização do serviço e a invocação do mesmo por parte dos recursos dos serviços web passa a trabalhar com o paradigma da orientação a serviços. Quando a aplicação solicitante recebe a resposta do serviço solicitado é necessário efetuar a transformação da informação de XML novamente para a linguagem entendida pela aplicação, processo de *unmarshal*, entrando novamente em ação todos os recursos do mundo OO, dependendo é claro da linguagem adotada.

Outra forma onde se pode comprovar claramente a importância da orientação a objetos no uso dos serviços web diz respeito a concepção das ferramentas utilizadas para a geração dos documentos WSDL, SOAP e UDDI necessário a disponibilização dos serviços-web na internet, ou seja os toolkits disponibilizados por inúmeras empresas de softwares. Esse processo pode ser realizado de forma manual, com a elaboração de cada um dos arquivos em editores puros. Entretanto, existem ferramentas que auxiliam em muito esse processo e, as principais delas, que oferecem a maior gama de recursos, foram desenvolvidas em linguagens totalmente orientadas a objetos, como é o caso das diversas API's fornecidas pela SUN e ferramentas específicas da IBM e da própria Microsoft.

Os serviços web são uma realidade e estão revolucionando não só a forma das pessoas interagirem com as aplicações, mas principalmente a forma das aplicações interagirem com as próprias aplicações. Isso permitirá sistemas realmente dinâmicos.

Analisando o avanço possibilitado pelos serviços web e algumas ponderações realizadas por pesquisadores da área de orientação a objetos, dentro os quais James Martim pode-se constar que praticamente tudo o que se pensou em nível de reusabilidade e interoperabilidade hoje já está ao alcance, falta apenas coloca-las em prática e difundir essa tecnologia em todo o mundo.

James Martin já afirmava em 1993 “estamos no final de uma era de software de fornecedor único. O software do futuro conterá componentes de milhares de fornecedores”. Os serviços web de certo modo levam a um sistema composto por diversos fornecedores, a exemplo do que pode ocorrer um serviços web de uma agência de turismo que se utilizará de softwares diversos de seus parceiros e dos recursos (serviços) oferecidos pelos mesmos.

Ainda prevendo como os softwares deveriam se comportar no futuro, entre tantas outras observações de James Martim, destacam-se as seguintes:

“Que existam padrões abertos, para permitir que os objetos se intercomuniquem”.

“Que os componentes residam em um repositório padrão. Na ausência de padrões internacionais abertos para repositórios de softwares padrões de fato de grandes fornecedores ou associações e confederações industriais definirão o metamodelo e as interfaces de usuários”.

“Que os protocolos especificados permitam a interação entre processos entre máquinas hospedeiras e através das redes”.

Diante dessas considerações realizadas há uma década é possível verificar que muitas coisas hoje são realidade dentro do uso dos serviços web. O paradigma de orientação a serviços, sem sombra de dúvidas veio somar ao paradigma da orientação a objetos e a evolução dos sistemas depende do casamento destes dois paradigmas. A grande vantagem de se trabalhar com ambos de forma conjunta é se possibilitar que tudo o que já foi desenvolvido até hoje seja reaproveitado, pois do contrário seria difícil convencer a grande maioria de empresas, tanto de tecnologia, como de bens e serviços a rever todo o investimento histórico realizado na área de tecnologia da informação.

A essência dos sistemas para o futuro não será a elaboração de novas maravilhas, mas sim o reuso e interoperação entre o novo e o que já esta consolidado, pois

a reusabilidade e a interoperabilidade são o carro chefe de muitos serviços e protocolos disponíveis na internet e que irão revolucionar a forma de comunicação do planeta.

10 CONCLUSÃO

Segundo Albert Einsten, a necessidade é o fator gerador ou motivador dos novos paradigmas. Na área de tecnologia da informação essa é uma verdade praticamente absoluta.

No decorrer do presente trabalho foi possível acompanhar a evolução dos paradigmas envolvidos com o desenvolvimento de software e as necessidades que levaram ao surgimento dos mesmos. Após um longo período de evolução dos softwares as necessidades latentes em todas as áreas envolvidas chamam-se interoperabilidade e reusabilidade, e, as pesquisas sobre as mesmas nortearam a realização desse trabalho.

Inicialmente foi realizado um estudo sobre o paradigma de orientação a objetos. Os conceitos envolvidos no mundo OO já se encontram bem difundidos em todas as áreas da informática, o que facilitou em certo modo a pesquisa e a comprovação das vantagens do uso desse paradigma, principalmente a da reusabilidade e modularidade, que tem como pilares a herança e o polimorfismo.

Na seqüência foram realizadas inúmeras leituras e comparações sobre objetos distribuídos e *middlewares* envolvidos na comunicação entre os sistemas distribuídos. Nesta categoria destacam-se o DCOM, o CORBA e o RMI, que numa evolução apresentam ainda uma solução CORBA/RMI e Java integrada. Foi possível constatar os enormes avanços registrados por essas tecnologias e certamente elas continuaram a ser utilizadas seja em aplicações locais ou distribuídas já implementadas, buscando, no entanto, a integração com a arquitetura orientada a serviços.

O passo seguinte foi à busca por informações e bibliografia sobre os serviços web e as tecnologias envolvidas, ou seja, HTTP, SOAP, XML, UDDI, WSDL, eb-XML, entre outros. Essa fase constitui-se na mais trabalhosa, devido ao fato de se constituírem em

recursos e tecnologias novas que, portanto, não apresentam muitas publicações, sendo a maior fonte de referências às normas ou ensaios e as respectivas documentações.

Toda a herança da evolução dos softwares em si, mais recentemente do paradigma da orientação a objetos e os recursos oferecidos pelos serviços web necessitam de um tradutor universal, esse tradutor universal é a XML.

Devido à dinamicidade com que esses padrões avançaram encontrou-se uma certa dificuldade me relação ao acompanhamento de novas ferramentas e dos padrões em si. Desde o início do trabalho até a sua conclusão novas versões de alguns dos protocolos foram homologadas e inúmeras iniciativas envolvendo as empresas de tecnologia ligadas à padronização dos serviços web foram tomadas, o que com certeza fez com que alguns dos recursos apresentados aqui já estejam aprimorados em alguns aspectos, como por exemplo, a questão da segurança dos serviços web.

Outra dificuldade encontrada diz respeito à quantidade de tecnologias e recursos envolvidos, desde o resgate dos sistemas distribuídos em si até os próprios serviços web. Muitas tecnologias acabavam puxando outras e o entendimento dos conceitos abordados se fazia necessário para a conclusão do todo, exigindo uma pesquisa maior também sobre os assuntos não diretamente citados no trabalho.

O estudo das tecnologias envolvidas na SOA, na OO, em especial a própria XML e a realização de outras pesquisas em nível acadêmico levou a comparações e constatações que permitiram atingir os objetivos deste trabalho.

10.1 Alcance dos Objetivos

Através das pesquisas realizadas e das diversas comparações entre o SOA e OO fica comprovado que a integração entre essas duas tecnologias interferem de forma direta e podem garantir a interoperabilidade e a reusabilidade de aplicações diversas.

Logicamente existem inúmeras barreiras a serem vencidas ou diminuídas, como questões que envolvem segurança nos seus mais diversos aspectos e, principalmente a vontade política de empresas de softwares a desenvolvedores, se bem que nos últimos tempos isso também tem melhorado. Nunca na história da informática se teve tantos recursos à disposição e principalmente nessas condições permitindo aproveitamento praticamente total do arcabouço de sistemas já existente. O que é preciso agora é que aos poucos os desenvolvedores e equipes de projetos mudem a sua forma de trabalhar,

iniciando o projeto pela modelagem das classes, componentes, aplicativos ou serviços web que são possíveis de serem reutilizados.

No formato XML são descritos os registros dos serviços web que se constituem na sua maioria em registros UDDI, os quais apresentam informações sobre os serviços web existentes. Uma dessas informações constitui-se na localização do arquivo WSDL, que descreve a forma como o serviço web deve ser solicitado, ou seja, no documento WSDL estão explicitados quais os métodos a serem utilizados para invocação e resposta de um serviço, bem como tipos e demais informações necessárias. O SOAP define um *framework*, aberto, no qual são apenas adicionadas as informações advindas do arquivo WSDL do serviço web que se deseja acessar. Além dessas tecnologias existem outras importantes iniciativas que estão buscando tratar problemas como segurança, autenticação, criptografia e principalmente integração de aplicações complexas que apresentam negociações entre vários serviços web e necessitam inclusive se controle de concorrência.

A comparação entre a arquitetura orientada a serviços e o paradigma da orientação a objetos foi realizado de forma detalhada no capítulo 09. A orientação a objetos fornece toda a estrutura para a sustentação das aplicações que serviram como base aos serviços web, uma vez que, no decorrer das pesquisas ficou comprovado o uso quase que total de ferramentas orientadas a serviço para criação de aplicações voltadas a tal finalidade, e a orientação a serviço prove o transporte da mensagem entre as diversas aplicações. Na realidade, os serviços web não se constituem de objetos em si, no sentido formal e clássico de orientação a objetos, no entanto é possível mapear objetos distribuídos neles e acessá-los de forma dinâmica com o uso de linguagens e *toolkits* totalmente orientados a objetos. Uma plataforma de objetos distribuídos é um ambiente de suporte essencial para a criação de uma camada de serviços web. A idéia fundamental deve ser a de que os objetos distribuídos são os blocos essenciais de construção de serviços escaláveis e de alta disponibilidade em uma empresa, e que serviços web são os pontos de acesso à utilização desses serviços, que permitem seu compartilhamento com sistemas internos de outras empresas.

Em relação às ferramentas disponíveis para manutenção em serviços web, também chamadas de *toolkits*, as mesmas são oferecidas por grandes empresas como a Sun Microsystems e a IBM, mas com certeza existem inúmeras ferramentas desenvolvidas por empresas de menor porte. No decorrer das pesquisas foram abordadas de forma superficial as seguintes ferramentas: IBM *Web Service Toolkit* (WSTK), o *Java Web Service Developer Pack* (JWSDP), W*eb Application and Service Plataform* (WASP), *CapeStudio*. O que chama atenção é que todas elas tem suporte ou uma relação direta com a linguagem

Java, comprovando mais uma vez a integração entre OO e SOA. Algumas dessas ferramentas já apresentam um conjunto de classes que permite a realização do acesso dinâmico aos serviços web. Além dessas ferramentas existem soluções que envolvem o desenvolvimento desde as aplicações até o processo de integração com os serviços web, dentre eles os mais importantes são o Visual.Net da Microsoft e o Java One da Sun, ambos utilizando-se de metodologias e linguagens orientadas a objetos.

O uso de padrões e, principalmente padrões abertos é uma realidade que acompanha muitas das mais significativas ações que deram certo na área de tecnologia. A Internet é o exemplo mais latente dessa constatação. A definição de uma pilha de protocolos padrão e aberta fez com que a Internet se espalhasse pelos quatro cantos do mundo. Dificilmente se encontra uma instituição que utiliza tecnologia da informação que não conheça ou utilize padrões TCP/IP HTTP e HTML. Outro exemplo que pode ser citado de forma extremamente marcante é a UML que esta sendo utilizada em nível universal para os mais diferentes tipos de projetos.

Como tudo indica os serviços web estão aos poucos se enquadrando nesse caminho e, talvez esse seja um dos aspectos que esta levando as pessoas envolvidas com a área de tecnologia a investir e acreditar na arquitetura orientada a serviços. Na realidade pode-se comprovar claramente uma junção de padrões abertos para a definição e uso dos serviços web em si. Pois desde os projetos realizados até sua execução sobre o protocolo http, em todos os momentos estão sendo utilizados importantes padrões do mundo tecnológico.

Diante do que foi exposto acima se pode concluir que hoje é possível se trabalhar de forma interoperável entre diferentes aplicações, se utilizando inclusive de aplicações já existentes ou outras formas de reusabilidade. O desenvolvimento de aplicações para a Internet não pode mais ser pensado sem o foco da interoperabilidade e da reusabilidade, pois com os recursos oferecidos pelos serviços web isso representara o mesmo que uma instituição com um conjunto de computadores e aplicações que não se comunicam.

O correto uso dos serviços web conduzira a uma nova forma de se pensar a tecnologia da informação, mais inteligente e mais otimizada, pois informações poderão ser simplesmente obtidas em forma de serviço, ao invés de ficarem armazenadas em cada aplicação de forma desconexa, como por exemplo, os dados cadastrais de qualquer indivíduo que, mediante a existência de um serviço web na Receita Federal podem ser acessados diretamente deste serviço, com todas as informações disponíveis.

10.2 Propostas para Trabalho Futuros

Sendo um assunto novo, o tema que envolve serviços web e em especial os assuntos abordados nesse trabalho oferecem inúmeras possibilidades de continuidade para estas pesquisas, principalmente em áreas que envolvam os serviços web complexos, as questões de segurança, ferramentas para desenvolvimento e gerenciamento. Além disso, como foi possível perceber ao longo do trabalho existem inúmeras tecnologias relacionadas, que foram abordadas aqui de forma superficial e que podem aprofundadas. Dentre tantos, alguns temas que poderão constituir-se em trabalhos de pesquisa são:

- Escolha do melhor serviço web mediante avaliação da qualidade de serviço oferecida (QoS).
- Evolução das técnicas que permitem controle de falhas e exceção nos serviços web Os serviços web complexos poderão utilizar-se de diversos outros serviços disponíveis na Internet. Para tanto faz-se necessário uma maior preocupação no que tange ao controle de falhas, tendo em vista a garantia da integridade dos dados e o processo de concorrência em si, principalmente quando operações entre os serviços são canceladas ou perdidas devido inclusive a problemas de conexão.
- Metodologia para gerenciar, monitorar e analisar o uso e a execução dos serviços web, avaliando status da aplicação, performance, uso de CPU, utilização de memória.
- Uso de serviços web na prestação de serviços.
- Segurança nos serviços web. Esse é um assunto que merece inúmeras pesquisas para que seja possível trabalhar com um nível de segurança que possibilite amenizar as inúmeras brechas de segurança que a Internet possui.
- Definição de segurança ENDtoEND com o uso de XML.
- Roteamento dinâmico de serviços web.
- Processamento transacional em diferentes camadas de negócios nos serviços web.
- Garantia de integridade nas transações em serviços web.

- Integração dinâmica entre serviços web.
- Gerenciamento de web serviços.
- Tecnologias e paradigmas envolvidos nos serviços web complexos.

Como o tema serviços web é novo, abrem-se a partir desse trabalho muitas oportunidades de pesquisas além das citadas acima. Com certeza novas tecnologias serão no futuro somadas aos serviços web, como por exemplo a utilização de dados complexos, o que levará a pesquisas mais aplicadas na área de banco de dados voltados para o uso de banco de dados OO para sistematização dos dados complexos e disponibilização dos mesmos através de serviços web.

O trabalho ora concluído representa uma pequena contribuição na pesquisa acadêmica voltada principalmente para a área de Internet.

A ciência tem evoluído muito e, a tecnologia de informação tem contribuído muito para essa evolução, com passos extremamente marcantes nesse processo.

As conquistas que hoje se presenciam se refletirão em conforto para muitos num futuro não muito distante.

Na área do trabalho aqui apresentado isso certamente poderá ser comprovado dentro de alguns anos quando a tecnologia dos serviços web estará presente em lares e empresas, embora muitas pessoas nem se quer ouvirão falar nas tecnologias envolvidas. Mas o processo é esse: é preciso as pessoas que criem, pesquisem e implantem as tecnologias para que outras simplesmente as usem.

No futuro talvez se possa presenciar as pessoas utilizando a tecnologia para que as suas compras sejam realizadas por sistemas de leitores automatizados direcionados a serviços web devidamente pesquisados na Internet que farão, por conseguinte a busca dos dados do cliente em outro serviço web, acionando outros serviços web para entrega respectivos débitos, de uma forma totalmente segura e precisa ou ainda através de comandos de voz e imagens que serão armazenados em bancos capazes de trabalhar transparentemente com dados complexos.

Toda essa evolução que se vivencia hoje e que o futuro reserva, é fruto das mãos dos seres humanos e, muitas vezes é necessário muito estudo e muita pesquisa para se evoluir em pequenas coisas.

O Brasil precisa olhar com mais atenção para a necessidade de formar pesquisadores, não pela simples arte de pesquisar, mas sim visando com as pesquisas contribuir para melhor a qualidade de vida do país e do mundo como um todo.

A capacidade e o interesse que as pessoas tem de aprender e construir precisam ser valorizadas, pois é dessa forma que as invenções se renovam e que a tecnologia avança.

Talvez toda essa gama de recursos que hoje os serviços web possibilitam sejam fruto de um trabalho iniciado há muito tempo, mais precisamente no ano de 1969, por Ed Mosher, Ray Lorie e Charles F. Goldfarb, que inventaram a primeira linguagem de marcação moderna, a *General Markup Language* (GML), da qual se originou a SGML e na sua evolução a HTML, que possibilitou a interconexão das informações no mundo e, mais recentemente a XML que esta se firmando como o tradutor universal de dados para aplicações distintas. Esse exemplo com certeza mostra a necessidade de se investir no hoje para colher os frutos amanhã.

REFERÊNCIAS

(01) Burbeck, S. **The Tao of e-business services – The evolution of Web applications into service-oriented components with Web services**. IBM Software Group, Oct 2000. Also available at:

(02) COMER, Douglas E.; STEVENS, David L. **Interligação em Rede com TCP/IP**, volume 2. Tradução Ana Maria Neto Guz. Rio de Janeiro; Campus, 1999.

(03) **Composição em WebFrameworks.** Capturado em 6 out. 2002. Online. Disponível na Internet http://www.inf.unisinos.br/~crespo/publicacoes.html

(04) DEITEL, II. M. **Java como programar** 3.ed. Porto Alegre : Bookman, 2002

(05) **About Eiffel Software**. Capturado em maio de 2002. Online. Disponível na Internet em http://www.eiffel.com/general/

(06) FOWLER, Martin, SCOTT, Kendall. **UML essencial, um breve guia para a linguagem-padrão de modelagem a objetos**. Price. 2. ed. Porto Alegre : Bookman, 2000

(07) MARTIN, James**. Princípios de análise e projetos baseados em objetos**. Rio de Janeiro, Editora Campus, 1994.

(08) Comitê gestor da Internet no Brasil. Online. Capturado em fevereiro de 2003. Disponível em http://www.cg.org.br

(09) **Frameworks orientados a objetos**. Capturado em 6 out. 2002. Online. Disponível na Internet http://www.revista.unicamp.br/infotec/informacao

(10) **Portal institucional da SUN**. Online. Capturado em maio de 2002. Disponível em http://sun.com

(11) **Integração, descoberta e descrição universal de web services**. Capturado em junho de 2002. Online. Disponível em http://www.uddi.org/

(12) **Portal da UML**. Online. Capturado em abril de 2002. Disponível em http://www.uml.org.

(13) **Estudo e utilização de frameworks orientados a objetos**. Capturado em setembro de 2002. Online. Disponível na internet em http://www.cic.unb.br/docentes/fernando/projetos

(14) **Conceito orientação a objetos**. Capturado em 13 out. 2002. Online.

Disponível na Internet http://ftp.unicamp.br/pub/apoio/treinamentos/OO

(15) **Portal do XML**. Capturado em abril de 2002. Online. Disponível em http::/www.xml.org

(16) **Documentação das normas que envolvem as tecnologias SOA**. Capturado em maio de 2002. Online. Disponível na Internet em http://www.w3c.org/2002/ws/

(17) **Portal institucional dos web services**. Capturado em julho de 2002. Online. Disponível em http://www.webservices.org/