UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE TECNOLOGIAS E GEOCIÊNCIAS

DEPARTAMENTO DE ELETRÔNICA E SISTEMAS

Pós-Graduação em Engenharia Elétrica

Divulgação do Orçamento Público via Web Services em Java

por

Jorge Abilio Abinader Neto

Dissertação de Mestrado

Recife – Fevereiro de 2004

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

JORGE ABILIO ABINADER NETO

Divulgação do Orçamento Público via Web Services em Java

Este trabalho foi apresentado à Pós-Graduação em Engenharia Elétrica do Centro de Tecnologias e Geociências da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica.

ORIENTADOR: Prof. Dr. Rafael Dueire Lins

Recife – Fevereiro de 2004

JORGE ABILIO ABINADER NETO

Divulgação do Orçamento Público via Web Services em Java

Aprovado em 20 de fevereiro de 2004

Dissertação apresentada como requisito parcial à obtenção do grau de mestre. Programa de Pós-Graduação em Engenharia Elétrica do Centro de Tecnologia e Geociências/Escola de Engenharia de Pernambuco, Universidade Federal de Pernambuco.

BANCA EXAMINADORA

Prof. Rafael Dueire Lins, PhD.
Orientador

Profa. Fernanda Maria Ribeiro de Alencar, PhD.

Prof. Ricardo Massa Ferreira Lima, PhD.

RESUMO

A complexidade e o crescimento da Internet fizeram com que vários modelos e propostas para a comunicação entre aplicações fossem apresentados como opção de computação distribuída neste ambiente. Porém, em todos estes faz-se necessário a introdução de requisitos próprios de cada modelo, a serem adotados por aqueles que desejam utilizá-los. Como alternativa, apresenta-se a tecnologia de Web Services, caracterizada pelo baixo acoplamento entre as partes e fundamentada em XML, HTTP e outros padrões já estabelecidos na Internet. Para aderir a esta tecnologia, a plataforma de desenvolvimento Java apresenta o software Java WSDP(Java Web Services Development Package) que possibilita o fornecimento e o consumo de Web Services a partir de programação Java. Nas redes metropolitanas do poder executivo brasileiro, ocorre situação semelhante a heterogeneidade de linguagens e plataformas, como encontramos na Internet. Neste contexto, este trabalho propõe a implementação de Web Service em Java para a divulgação do orçamento público, como solução para integrar órgãos participantes das redes metropolitanas e disponibilizar tais informações para acesso público. Apresentamos estudo de caso relativo ao contexto anteriormente citado e implementamos protótipo para fornecimento de Web Services, bem como aplicação web para consumo e divulgação das informações obtidas.

ABSTRACT

The Internet's growing and complexity enable many models and proposals for communications between applications in this environment. However, every these way need to introduce ourselves restrictions that must be use by the parts. As an option, there is Web Services technology that provide loosely-coupled how it principal characteristic, is based on XML, HTTP and others Internet stable standards. Java development platform adhere on it presenting the Java WSDP that possibility supply and consumer of Web Services by the Java programming way. The Brazilian government's metropolitan networks is like Internet at the heterogeneous languages and platforms. This work proposal is to implement a Web Services using Java to publish the public budget execution in this scenario, like a solution to integrate the entities of government that are connect by the metropolitan networks and left this public budget information to public access by Web Services. We offer the case study for the context before mention and implemented the prototype to supply Web Services and the Web application to use it and display the information.

SUMÁRIO

LISTA DE ABREVIATURAS

ACID	Atomic, Consistent, Isolated and Durable
ANSI	American National Standards Institute
API	Application Program Interface
ASCII	American Standard Code forInformation Interchange
B2B	Business to Business
BEEP	Block Extensible Exchange Protocol
BPSS	Business Process Service Specification
BTP	Business Transaction Protocol
CCITT	Consultative Committee for International Telegraphy and Telephony)
CNPJ	Cadastro Nacional de Pessoa Jurídica
COM	Component Object Model
CORBA	Commom Object Request Broker Architecture
CPP/CPA	Colaborative Protocol Profile/Colaborative Protocol Agreement
DCOM	Distribuited Common Object Model
DIME	Direct Internet Message Encapsulation
DSML	Directory Services Markup Language
DTD	Data Type Definitions
EBXML	Eletronic Business XML
EDI	Eletronic Data Interchange
EIS	Enterprise Information Systems
EJB	Enterprise Java Beans
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
GCI	Global Commerce Initiative
GUI	Graphical User Interface
HL 7	Health Level 7
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPR	HTTP RELIABLE
HTTPS	HTTP Secure

IBM	International Business Machine
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IIOP	Internet Inter-ORB Protocol
IMAP	Internet Message Acess Protocol
ISO	International Standard Organization
J2EE	Java 2 Enterprise Edition
JAX B	Java Architecture for XML Binding
JAX M	Java API for XML Messaging
JAX P	Java API for XML Processing
JAX RPC	Java API for XML RPC
JCP	Java Community Process
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
JVM	Java Virtual Machine
JWSDP	Java Web Service Development Package
LDAP	Lightweight Directory Access Protocol
MIME	Multipurpose Internet Mail Extensions
MOM	Message-Oriented Midleware
MSH	Messaging Service Handler
MVC	Model View Controler
OAGI	Open Application Group, Inc.
OASIS	Organization for Advanced Structured Information Standards
OLE	Object Linking and Embedding
OMG	Object Management Group
ORB	Object Request Broker
OTA	Open Travel Alliance
PDA	Personal Digital Assistant
PKI	Public Key Infrastructure
POP3	Post Office Protocol 3
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAAJ	Soap with Attachment API for Java
SAML	Security Assertions Markup Language
SGML	Standard Generalized Markup Language
SMTP	Simple Transfer Message Protocol

SOAP	Single Object Acess Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UBR	UDDI Business Registry
UDDI	Universal Description, Dicovery Interface
UN/CEFACT	United Nations Center for Trade Facilitation and Eletronic Business
URL	Universal Resource Locator
W3C	World Wide Web Consortium
WSCI	Web Services Choreography Interface
WSDL	Web Services Description Language
WUST	WSDL-UDDI-SOAP Tecnologies
XACML	Extensible Acess Control Markup Language
X-DSIG	XML Signature
X-KISS	XML Key Information Service Specification
XKMS	XML Key Management System
X-KRSS	XML Key Registration Service Specification
XML	Extended Markup Language
XML ENC	XML Encription
XSL	Extensible Stylesheet Language

LISTA DE FIGURAS

CAPÍTULO 1 - Introdução

1.1 Introdução

A democratização dos meios de comunicação e a presença da Internet no cotidiano da sociedade, produziram sobre o Estado Brasileiro demandas e desafios para a disponibilização de informações públicas na web. Nestas demandas estão incluídos os serviços de consulta e interação via navegador (browser), aplicações de troca de arquivos e outras que utilizam a infra-estrutura de Internet como ambiente nativo.

Muitas tecnologias para construção de aplicações na Internet têm surgido e evoluído nos últimos anos. A orientação a objetos marca o início de um novo modelo de projeto e programação que reforçado pelo amadurecimento da linguagem Java e suas tecnologias deram origem ao modo e forma de construir sistemas em Java denominada J2EE. A linguagem XML também contribui de modo fundamental para a interoperabilidade entre sistemas e aplicações heterogêneas neste ambiente.

As aplicações de Internet nas quais o usuário obtém o que deseja a partir de uma tela de opções tornam-se cada vez mais sofisticadas e elegantes. Porém, com o emprego maciço da web, descobriu-se que esta poderosa ferramenta de comunicação poderia ser usada também para transações no atacado entre corporações, governos e seus parceiros.

Porém a interação corporativa entre aplicações, via Internet, exige esforço extra de adequação de ambos os lados de modo que as mesmas sejam compatíveis em alto grau de sintaxe e semântica. Neste contexto para que uma aplicação de uma corporação/governo possa interagir com outra aplicação de um parceiro são necessárias adaptações que tornam o relacionamento altamente interdependente e direcionado, de tal modo que no caso de mudanças ou a não disponibilidade de um dos lados, o outro ficará sem possibilidade de funcionar adequadamente.

Neste estudo de caso, apresentamos proposta para a aplicação de Web Services em Java com o objetivo de possibilitar a interação entre aplicações governamentais e demais entes da sociedade, com baixo grau de interdependência (acoplamento) e pouco esforço de adequação (interfaces para acesso a informações disponíveis em XML). Como base da pesquisa tomaremos a execução do orçamento oferecendo protótipo de Web Services em Java que possibilite a localização, a conexão e o uso na obtenção de informações sobre este aspecto do governo.

1.2 Motivação

As aplicações de Internet para a interação com o usuário crescem rapidamente. Muitas linguagens e opções de tecnologia são utilizadas, pois a interação com o usuário passa pelo entendimento e capacidade de utilização de uma interface gráfica enriquecida por recursos de cor, animação etc. Temos neste caso a inteligência humana capaz de interagir e "navegar" pelo programa para obter a informação desejada.

Aplicativos e sistemas corporativos precisam comunicar-se, via Internet, trocando dados mutuamente. O modelo de aplicação anteriormente citado, demonstra-se inadequado para o relacionamento entre parceiros de negócios que precisam cooperar entre si trocando, de modo automatizado e confiável, grandes volumes de informações.

Isto posto, ao cogitarmos a adoção deste modelo de aplicação para tal relacionamento, estaríamos retornando ao velho modelo do mainframe, só que atualizado para a tecnologia de Internet. Na qual cada parceiro, por sua vez, disponibilizaria a sua aplicação para o outro "digitar" e obter o que fosse necessário para a realização de transações e negócios.

O EDI (Eletronic Data Interchange)[1] foi adequado para um cenário bastante diferente, pois exige soluções altamente personalizadas, com alto custo de implementação e uso, tecnologia de comunicação de valor elevado, tornando-se inviável para grande maioria de potenciais usuários, além de tornar-se um desafio de implementação, devido a sua complexidade.

[1] Segundo [13], EDI constitui-se no conjunto de especificações para transações de negócios realizadas eletronicamente, por exemplo em uma rede de computadores. O padrão EDI especifica o tipo de informação que deve estar disponível ou deve ser trocada entre as possíveis transações a serem realizadas. O padrão também especifica o formato que esta informação deve ter. EDI é descrito nos seguintes documentos oficiais : recomendação ISO 9735; documentos ANSI X12.3 e X12.22 e recomendação CCITT x.435.

Entretanto, a Internet pode ser usada para que governos e empresas interajam entre si, reduzindo custos e de modo a possibilitar maior facilidade e rapidez na troca de informações. A demanda atual é a disponibilização de meios para que as aplicações de governo e corporativas obtenham na web as informações de que necessitam para consolidar resultados.

Para que duas aplicações ou sistemas corporativos passem a trocar informações via Internet, faz-se necessário que os parceiros envolvidos estabeleçam acordo de aderência mútua em torno de um modelo, adaptando-se ao mesmo para concretização da comunicação intersistemas.

Este processo de adaptação, em alguns casos, exige exaustivo esforço colaborativo exercido através de intermináveis reuniões, contatos via e-mail, testes e outros expedientes da interação humana prévio.

A interação entre aplicações do governo, foco deste trabalho, deve ocorrer de modo simplificado, padronizado e principalmente não proprietário. Existem, atualmente, algumas soluções que podem permitir o alcance deste objetivo. Entre estas a tecnologia de Web Services em Java que permite a interação entre aplicações sem que haja o contato humano.

Neste contexto, temos no Brasil, a visão fazendocêntrica[2] de que somente a Secretária de Fazenda ou Receita, deve possuir e centralizar informações relativas a execução orçamentária do Estado (nas esferas municipal, estadual e federal).

Porém, ocorre que, com a modernização, capacitação e descentralização das ações na administração pública, a demanda para a disponibilização de informações relativas a execução do orçamento nos demais órgãos do governo torna-se crescente e indispensável.

Em decorrência da cobrança social intensa, será impossível ao gestor público de segmentos críticos (como por exemplo: saúde, educação, assistência social, segurança e outros) administrar sua pasta de forma adequada sem acesso flexível ao conjunto total das informações de execução orçamentária relacionadas a atividade fim do segmento pelo qual está responsável.

Considerando que na grande maioria dos casos, os governos no Brasil omitiram-se quanto a padronização e a adoção de políticas de orientação para investimento e custeio das estruturas

[2] Refere-se a sistemas computação construídos para administração fazendária que nasceram a partir do órgão arrecadador e administrador do orçamento público. Foram impostos aos demais órgãos, considerados de periferia e menor importância, tendo que adaptarem-se as exigências orçamentárias, contábeis, financeiras e muitas outras para que a Lei 4.320/64 fosse atendida.

de informática. Permitindo que cada gestão, gestor público ou entidade adota-se suas próprias soluções de tecnologia e construísse seus próprios acervos de dados (chamados por vezes de bancos de dados).

Esta omissão propiciou condições ideais para a instalação do caos que poderíamos chamar, modernamente, de heterogeneidade. O que para alguns pode parecer um desastre, para a tecnologia de Web Services torna-se oportunidade.

Assim a tecnologia de Web Services, aliada aos padrões estabelecidos da Internet, pode fazer com que os dados fluam entre as várias instâncias de governo, desde que haja esforço concentrado para que a semântica destes dados possa ser aglutinada e transformada em informação útil para o conjunto de entidades envolvidas e empenhadas no aumento da qualidade da administração pública.

Mas não há otimismo idealista, a integração de acervos de dados estanques, situados em órgãos, plataformas e sistemas diferentes, empregando-se Web Services em Java, apresenta-se como desafio complexo.

Ao considerarmos esta possibilidade, devemos contabilizar a necessidade da manutenção de dados legados, produzidos e mantidos por tecnologias, muitas vezes em desuso, que devem ser consultados como subsídio a decisões do presente e na prevenção do futuro. A integração e a disponibilização destas informações, via Web Services em Java, constitui-se uma das principais aplicações desta tecnologia a nível endógeno nas esferas governamentais.

No caso específico da execução do orçamento, informações de caráter público disponibilizadas de modo democrático na Internet que possam ser acessadas e usadas por programas em qualquer linguagem, sistema operacional ou plataforma de processador, podem produzir benefícios concretos para toda a sociedade. Um dos benefícios mais imediatos é a transparência fiscal.

Ainda na esfera dos poderes constituídos, Tribunais de Contas poderiam obter grande facilidade no exercício de suas funções através do acesso a Web Services que fornecessem informações de modo automático sobre a execução orçamentária e arrecadação. Pois poderiam exercer nova modalidade de auditoria a qual seguiria as ações do governo quase que em tempo real.

O emprego de Web Services em Java para a disponibilização de informações sobre a execução do orçamento, reúne tecnologias necessárias para tornar estas informações acessíveis e utilizáveis por grande parte de potenciais clientes na sociedade.

Através da Secretaria de Fazenda da Receita Federal o Governo do Brasil transformou a Internet no instrumento preferencial de interação com base tributária (contribuintes do Imposto de Renda). Neste cenário a disponibilização de informações relativas as ações econômicas, financeiras e sociais através de Web Services em Java, pode tornar-se, no futuro próximo, fator diferencial na atração de investimentos.

1.3 Objetivo

Este trabalho possui como objetivo geral estudar e documentar o estado da arte de Web Services em Java, analisando sua funcionalidade e aplicação, bem como aspectos fundamentais de tecnologias base para Web Services, tais como : a linguagem XML, protocolo SOAP, UDDI e WSDL.

Como cenário de pesquisa tomaremos sistema cliente/servidor baseado em banco de dados relacional e implementado em rede local. Este sistema é real e tem como meta o gerenciamento da execução orçamentária municipal em seus aspectos financeiro e contábil, de acordo com o determinado na Lei 4.320/64.

O objetivo específico é estudar aspectos fundamentais a serem considerados quando do emprego de Web Services em Java para :

- divulgar informações públicas na Internet baseado em modelo com baixo acoplamento entre órgãos do governo, proporcionando maior integração administrativa;
- propor o emprego de Web Services em Java como alternativa de integração de informações (sistemas) entre aplicações do governo implementadas em plataformas e linguagens heterogêneas.

Foi implementado protótipo que aborda duas perspectivas de Web Services em Java. Como cliente, acessando e utilizando informações disponibilizadas e como servidor, publicando Web Service para uso de outros órgãos do governo que necessitem das informações disponibilizadas pelo Web Services para sua funcionalidade cotidiana.

O emprego da linguagem Java e suas APIs será estudado e documentado de modo a permitir a implementação e otimização do protótipo proposto.

1.4 Organização da Dissertação

Os Capítulos que compõe este estudo foram divididos segundo a abrangência, de modo a permitir que a condução do assunto recebesse o embasamento teórico adequado a medida que a pesquisa fosse ganhando volume e reunindo pontos relevantes e necessários ao experimento prático ou seja a implementação do protótipo. Assim a estrutura adotada inicia por este Capítulo o qual apresenta introdução, motivação, objetivos.

Web Services constituem-se o ponto central do trabalho, sua aplicação em um estudo de caso prático, tem o objetivo de demonstrar, concretamente como podem ser utilizados para interferir positivamente no uso de computadores interligados a Internet. No Capítulo 2 posicionamos esta tecnologia no contexto das opções e em relação a outras tecnologias existentes para o desenvolvimento de aplicações distribuídas. Damos ênfase ao modelo de aplicação distribuída de maior capilaridade na Internet e de como a tecnologia de Web Services adapta-se a este modelo, retirando proveito para tornar-se aderente ao ambiente de Web. O Capítulo 2 encerra-se com a discussão de cada um dos fundamentos da tecnologia de Web Services de como funcionam, se integram e estão interligados aos conceitos relativos a SOAP, WSDL e UDDI, sem entretanto entrarmos em minúcias e detalhes tecnológicos sobre tais fundamentos.

Oferecemos no Capítulo 3 os fundamentos sobre as três tecnologias fundamentais que compõe a tecnologia de Web Services. Iniciamos porém, a discussão sobre a base da interoperabilidade, a linguagem XML que oferece o substrato para que possamos erguer as demais tecnologias que compõe a pilha de tecnologias Web Services, como citado em [10]. As tecnologias básicas que constituem Web Services quais sejam SOAP, WSDL e UDDI, são discutidas detalhadamente, onde os principais aspectos são abordados e oferecidos como fundamento necessário para a implementação de Web Services, a este grupo de tecnologias que [11] denomina de WUST acrônimo para WSDL-UDDI-SOAP e o T é de Tecnologies, segundo este autor Web Services pode ter várias definições, mas todas elas passaram por estas quatro letras. O Capítulo 3 é completado por citações de tecnologias importantes que agregam valor a Web Services, ampliando o escopo de aplicação.

A implementação, disponibilização e uso de Web Services pode ser executada por qualquer conjunto de artefatos que sejam aderentes aos padrões estabelecidos da Internet [3]. No Capítulo 4 oferecemos a visão da tecnologia de Web Services sob os aspectos relevantes da tecnologia Java, elaborados para que a mesma possa aderir a tecnologia de Web Services e contribuir, fortalecendo

conceitos Java, muito anteriores ao surgimento de Web Services, mas que mantém grande compatibilidade entre si como é o caso da portabilidade. Deste modo percorremos a tecnologia Java em quatro aspectos fundamentais representados por suas APIs JAX P, responsável pelo processamento de parsing, JAX RPC desempenha a função de disponibilizar Web Services RPC, JAX M tem como função permitir a implementação de Web Services baseados em comunicação orientada a mensagens e JAX R que propõe e executa padrão para a interação através da manutenção e consulta a registros de Web Services como os conhecidos padrões UDDI e ebXML.

Para demonstrar como a tecnologia de Web Services, em Java, constitui-se proposta concreta e factível para a divulgação do orçamento público, reunimos no Capítulo 5, informações necessárias sobre o cenário de aplicação deste estudo de caso, a execução do Orçamento Público. Selecionamos suas particularidades e capturamos, sob a ótica do emprego de Web Services em Java, a dinâmica operacional que esse ente administrativo possui. Definimos concretamente como a tecnologia de Web Services poderá atuar sobre o Orçamento Público no que diz respeito a sua divulgação.. Limitamos de modo formal o problema e apresentamos a solução a ser construída como protótipo proposto.

Partindo do cenário estabelecido no Capítulo 5, nossa proposta para o conteúdo do Capítulo 6, é a descrição do ciclo de construção de dois softwares que desempenham as funções de produtor de Web Services para a divulgação do orçamento público e de consumidor deste Web Services. Ambos construídos com a tecnologia Java. Além disso citamos o software Java a partir do qual desenvolvemos os protótipos bem como os softwares utilizados como apoio ao desenvolvimento. Encerramos o Capítulo 6, com uma lista de pontos que devem ser considerados quando da aplicação prática e em escala industrial das tecnologias de Web Services e Java no cenário delineado.

O encerramento deste trabalho, oferecido no Capítulo 7, constitui-se de três partes distintas . A primeira é composta de nossas considerações finais, que resumem o desenvolvimento do mesmo, relacionando e interligando fatos apresentados nos Capítulos anteriores e a implementação do software demonstrado. Na segunda parte, destacamos alguns pontos relevantes, considerados contribuições efetivas, que poderão ser empregados como referência para outros estudos na mesma linha. A terceira parte, compõe-se de algumas sugestões para a expansão deste estudo de caso, em trabalhos futuros, e de tópicos coletados durante a pesquisa que podem ser adotados como temas de novos trabalhos, relacionados a Web Services e a computação distribuída para aplicativos que utilizam a Internet como meio.

Capítulo 2 – Web Services

2.1 - Introdução

Web Services surgiram como conseqüência natural da utilização da Internet. Alguns consideram esta utilização massificada como um processo que produz a evolução[4] deste meio de comunicação entre pessoas e também como grande rede de computadores a qual naturalmente levou a possibilidade de se escrever aplicações e disponibiliza-las ao público em grande escala.

Inicialmente a Internet era constituída de páginas estáticas com informações interligadas, consultadas por pessoas com a utilização de programas chamados navegadores web, mais tarde popularizados com o nome de "browser", palavra em inglês que designa o mesmo tipo de programa. O conteúdo destas páginas só era possível de alteração quando a pessoa que mantinha a página realizava alterações. Estávamos na pré-história da Internet.

A evolução seguinte da Internet fez com que as páginas disponibilizadas se tornassem capazes de interagir, de acionar programas produtores de informações dinâmicas, provenientes de bancos de dados e outras fontes. Além dessas dinâmica, foi possível ao consumidor de conteúdos da Internet inserir, alterar e excluir informações, que em alguns casos também eram disponibilizadas para a consulta de terceiros.

Nesta fase de evolução, criou-se o modelo de aplicações distribuídas para Internet. Propiciando o fortalecimento de uma infra-estrutura de padrões, os quais incentivaram o aparecimento de várias soluções para a elaboração de aplicações destinadas ao cliente comum : navegador web (browser) com suas capacidades limitadas funcionando sobre as camadas HTML (Hypertext Markup Language) e HTTP (Hypertext Transfer Protocol)[2].

O navegador web (browser) tornou-se o cliente universal da Internet[12] para ser usado por seres humanos, como um ambiente onde se constrói interface com o usuário. Este fato provocou o surgimento de várias soluções, no lado servidor, para a construção de aplicações, que são capazes de extrair dados de várias fontes e disponibilizá-los através deste cliente.

Soluções foram elaboradas anteriormente, porém apresentaram-se de alto custo, proprietárias e complexas, características que eliminavam grande capilaridade de potenciais empresas que desejassem utilizar a Internet como meio de transação comercial[5].

Devido a ausência de padrões vários parceiros de negócios passaram a interagir na Internet através de soluções proprietárias. As quais produzem dependência de seus fabricantes. Criou-se então a demanda reprimida de negócios que poderiam ser realizados na Internet. Essa demanda pode ser definida, de modo simples, como a necessidade apresentada por negócios comuns, empresas de todos os tamanhos, em trocar informações entre sistemas diferentes que pudessem se comunicar através de padrões simples e públicos. A resposta para esta demanda é o que propõe a tecnologia de Web Services e a linguagem XML (Extended Markup Language) que pode ser considerado onipresente na Internet, simples, genérica, penetrante, barata e de fácil publicação[5].

Deste modo podemos oferecer duas visões dessa tecnologia que refletem os pontos de vista técnico e conceitual : do ponto de vista técnico Web Services constituem-se em software de baixo acoplamento, reusáveis, componentes feitos para ser facilmente acessados pela Internet. Na ótica conceitual o emprego de Web Services representa um modo para integrar tarefas que compõem um processo de negócio através da Internet, em uma cadeia de valor na qual procedimentos estão interligados e são interdependentes para atingir um resultado concreto final[5].

O conceito forte de autocontidos e modulares, faz com que os Web Services disponibilizem uma interface padrão que pode ser acessada pela Internet como único meio de interagir com estes. Desse modo um Web Service é uma aplicação que dispõe e publica uma API (Application Program Interface) na Web. Esta API suporta a comunicação programa-para-programa, o que permite que aplicações se comuniquem usando XML através da própria Web[2]. Temos aqui a simplicidade da tecnologia, pois existe o serviço que pode ser acessado pela Internet como se fosse uma caixa preta, cabendo ao cliente apenas saber que serviço deseja e como empregar tal serviço[5].

2.2 - Modelo de computação distribuída Web

O modelo de aplicação para Web baseado na interação humana, navegador web (browser) e servidor, constituiu-se no modelo de aplicação distribuída de maior adoção na Internet, superando modelos baseados em CORBA, DCOM e RPC[1]. Entre as principais características que fazem com este modelo seja amplamente adotado podemos citar :

- interação simples entre clientes (navegadores web) e servidores que trocam mensagens do tipo MIME, sendo que a semântica da mensagem pode ser modificada através do uso de cabeçalhos;
- destino da mensagem é especificado indiretamente com o uso de URL (Universal Resource Loader), propiciando a implementação de balanceamento de carga e controle de sessão, entre outras características necessárias a aplicações distribuídas;
- simplicidade para o acréscimo de recursos e novos membros no uso da aplicação tanto de clientes como de servidores, sendo necessários apenas um registro de DNS (Domain Naming System[3]) o que é natural no contexto da Internet;
- padrões bem estabelecidos e formados de domínio público, amplamente testados como HTTP e HTML;
- clientes de baixa complexidade e disponíveis na grande maioria de plataformas computacionais : navegadores web (browsers);
- todas ações ocorrem de forma descentralizada sem a necessidade de uma coordenação central o que fornece alto grau de interoperabilidade, escalabilidade e flexibilidade de conexão para uso de aplicações.

Assim, dois pontos principais precisam ser enfatizados : o baixo acoplamento entre cliente (navegadores web)/servidores e a interação humana com a interface do sistema disponibilizado. Como baixo acoplamento temos o fato de o cliente (navegadores web) endereçar a URL do servidor e este, quando em disponibilidade, oferecer o serviço, a aplicação desejada.

No caso da interface humana, esses sistemas são construídos para que o ser humano, interaja com o mesmo e obtenha o que deseja. Essas duas premissas aplicadas a esse modelo de aplicação distribuída na Internet o tornam de grande aceitação.

[3] DNS é um serviço de nomeação distribuída usado na Internet. O DNS tem a função de prover endereços IP para os computadores fornecendo nomes de domínios[13].

Considerações como o fato de haver total desacoplamento (cliente e servidor necessitam estar registrados em algum tipo de DNS na Internet) podem ser concretamente aferidas em ambos os lados do arranjo navegador web (browser) e servidor web. Pois para participar desse modelo o cliente (navegador web) deve ser capaz de localizar a URL do servidor desejado e possuir compatibilidade mínima com os padrões de HTTP, HTML não importando em que computador esteja, qual o microprocessador utilizado, qual o sistema operacional ou qual o idioma (já que existem esforços grandes para que a internacionalização se torne uma função comum neste tipo de aplicação).

No lado servidor, a princípio, necessita-se registro de DNS e da capacidade de responder a solicitações HTTP e idealmente em HTML, padrões já estabelecidos. Com essas habilidades não importa ao servidor que sistema operacional, processador, linguagem de programação ou banco de dados ele esteja empregando, as informações enviadas chegarão ao cliente e serão entendidas.

As possíveis diferenças entre cliente e servidor no modelo de aplicação distribuído amplamente adotado na Internet e aqui, resumidamente explicado, ficam completamente "empacotadas" e compatibilizadas sob três camadas básicas de software que se tornaram padrões de fato : TCP/IP protocolo de rede, HTTP protocolo de apresentação e HTML linguagem de hipertexto que é utilizada para descrever informações apresentadas no lado cliente. Temos assim estabelecido um contexto que desenvolveu-se com o uso massificado e a busca contínua de se disponibilizar aplicações e serviços para seres humanos interagirem, empregando como meio de comunicação a Internet. A Figura 2.2-1, a seguir, indica como estes padrões se estabeleceram. Do lado cliente temos as 3 camadas que servem para que a informação seja acessada. No lado servidor temos a informação produzida em HTML, depois transportada pelo HTTP e finalmente colocada na rede pelo TCP/IP que a divide em pacotes e entrega para a rede. No outro lado, o navegador web (browse) contém a mesma pilha de protocolos padrões, recebendo a informação e apresentado-a em HTML.

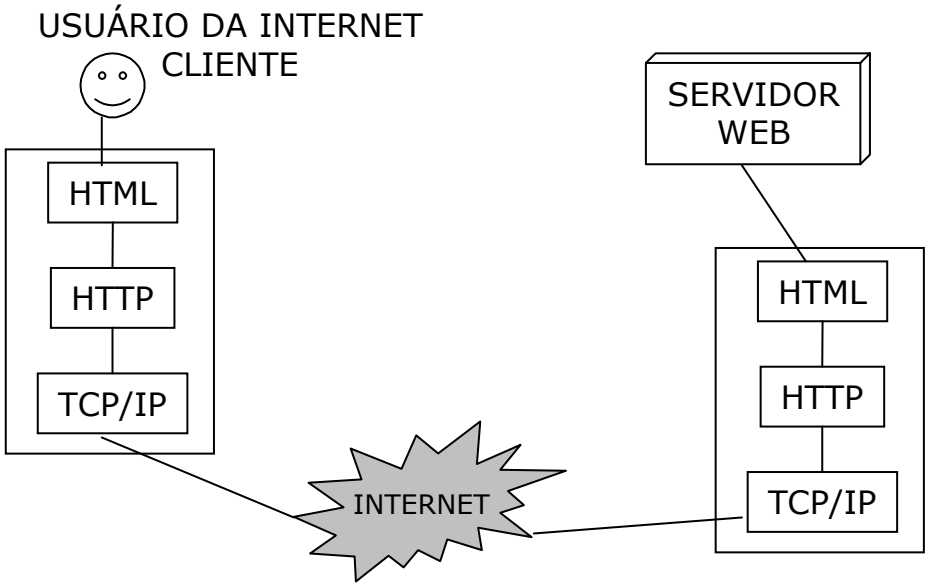


Fig. 2.2-1 ambiente padrão de Internet.

Esse cenário da Internet, indicado na Figura 2.2-1, chegou ao amadurecimento e a sua evolução natural foi a utilização da bem sedimentada infra-estrutura para usos mais sofisticados e complexos como por exemplo aplicações baseadas em Web nas quais programas interagissem com programas.

2.3 - Modelos de computação distribuída

Inicialmente os computadores de grande porte (mainframe) eram considerados soluções mais adequadas para a execução de aplicações distribuídas de grande escala e alto volume de processamento de dados e usuários[10]. Entretanto, o surgimento dos computadores pessoais e a sua interligação em rede, possibilitou o aparecimento de outras soluções possíveis para os sistemas distribuídos [31] .

O fortalecimento e o uso disseminado das redes de computadores (pessoais e de grande porte) como recurso computacional distribuído que poderia ser maximizado se pudesse vir a ser utilizado de modo coordenado, fez com que a computação em rede ganhasse importância e disponibilizasse as chamadas remotas a procedimentos (RPC de remote procedure calls) sobre o protocolo de rede denominado TCP/IP (Transmission Control Protocol/Internet Protocol) que foram amplamente aceitas como modo de comunicação entre aplicações[10].

Aplicações funcionando sobre grande variedade de plataformas de hardware, sistemas operacionais distintos e em redes de computadores diferentes, fizeram com que surgisse a demanda natural de compartilhar dados e integrarem-se em processos. Essa demanda produziu o conceito de aplicações de computação distribuída[10]. Assim, uma definição possível para computação distribuída pode ser, "Computação distribuída é uma espécie de computação na qual diferentes componentes de uma aplicação podem estar localizados em diferentes computadores conectados em rede"[10]. Tomando como base a Internet, apresentamos na Figura 2.3-2, o modelo de computação distribuída aqui considerado.

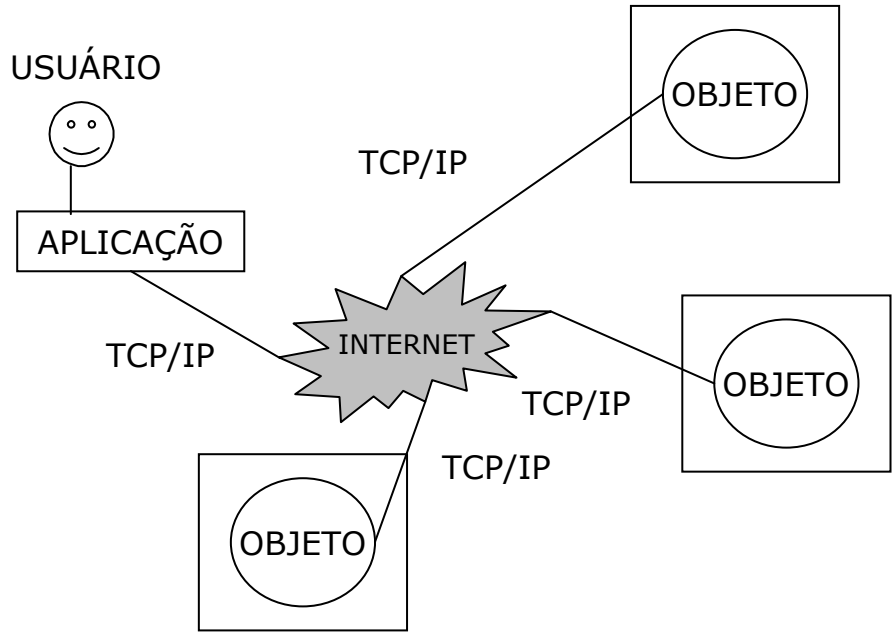


Fig. 2.3-2 modelo de aplicação distribuída na Internet.

No modelo da Figura 2.3-2, representamos de modo simplificado, como a computação distribuída provê a infra-estrutura que possibilita a aplicação o acesso a objetos funcionais que podem estar localizados (hospedados) em qualquer lugar na rede (no caso da figura a Internet) . A localização dos objetos é transparente para a aplicação e os mesmos fornecem poder de processamento e funcionalidade como se estivessem presentes no mesmo computador da aplicação que os invoca.

O ambiente de computação distribuída agrega muitas vantagens em relação ao ambiente de computação tradicional (chamado em inglês standalone, no qual um computador funciona sozinho rodando programas). Entre essas vantagens podemos citar :

- alto desempenho – aplicações podem executar em paralelo e distribuir a carga de processamento entre vários servidores;

- colaboração – muitas aplicações podem ser conectadas através de padrões a mecanismos de computação distribuída;
- alta confiabilidade e disponibilidade – aplicações e servidores podem ser colocados em conjuntos redundantes (em inglês clusters, onde um conjunto redundante de computadores responde de modo ordenado a uma solicitação, dividindo carga entre si de modo que se um vier a falhar, haverão outros capazes de responder a solicitação feita ao que falhou);
- escalabilidade – pode ser alcançada disponibilizando-se componentes distribuídos reutilizáveis em servidores poderosos;
- extensibilidade – pode ser executada através da capacidade dinâmica de configuração e reconfiguração da uma aplicação distribuída entre os recursos disponíveis na rede;
- alta produtividade e curto ciclo de desenvolvimento – quebrando a grande aplicação em pequenos componentes que podem ser desenvolvidos por pequenas equipes de forma isolada;
- reutilização – componentes distribuídos podem executar vários serviços, os quais, potencialmente, podem ser empregados por vários clientes em diversas aplicações. Isto economiza esforço repetitivo de desenvolvimento e aumenta a interoperabilidade entre componentes;
- custo reduzido – como o modelo possibilita alto grau de reutilização de componentes que podem ser usados (acessados) através da rede, reduções significativas nos custos de desenvolvimento podem ser atingidas[10].

A computação distribuída mudou o forma tradicional de programação de aplicações em rede ao possibilitar que objetos com semântica bem definida sejam compartilhados através da rede por programas escritos em linguagens diferentes como C, C++ e Java[40].

2.3.1 – Modelo de aplicação cliente-servidor

Os anos iniciais das aplicações de negócios distribuídas foram dominados pelo modelo de duas camadas[10]. Nesse modelo de arquitetura a camada mais próxima do usuário, chamada cliente, era responsável pela interface com o usuário e também pela lógica de negócio, funcionava nos computadores que ficavam nas mesas de trabalho dos usuários interligados a rede.

A segunda camada tinha a função de gerenciar a organização e o armazenamento de dados da aplicação, funcionava em um computador central da rede, era denominado de servidor. A esse tipo de arranjo e similares, se chamou de modelo cliente-servidor[10].

Genericamente, o servidor se constituía em um computador robusto funcionando como servidor de banco de dados com a responsabilidade de manter organizada a aplicação, armazenar os dados por ela processados e transferir, o mais rapidamente possível, esses dados para a aplicação e vice-versa.

O cliente fornecia interface gráfica com o usuário e processava a grande maioria dos requisitos necessários ao negócio.

Este modelo caracteriza-se por forte acoplamento entre a interface do usuário, a lógica de negócio e o servidor de banco de dados[10]. Tornou-se bastante popular e foi adotado pelos sistemas corporativos, pacotes de softwares aplicativos vendidos como soluções prontas denominados de ERP (Enterprise Resource Planning). A Figura 2.3.1-3 apresenta modelo típico da arquitetura cliente-servidor, no qual podemos observar que vários computadores dos usuários acessam o servidor de banco de dados central através da rede.

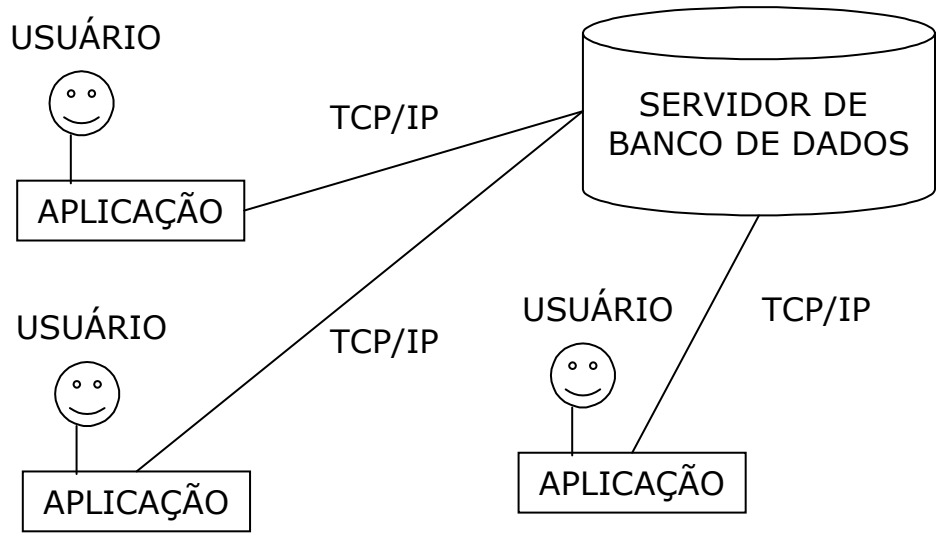


Fig. 2.3.1-3 modelo de aplicação cliente/servidor.

O modelo cliente-servidor possui algumas limitações que o tornam inadequado para a adoção no ambiente de Internet, como por exemplo : o processamento complexo das regras de negócio exige clientes robustos; a segurança é frágil, pois as regras de negócio residem no cliente, tornando sua preservação difícil; a largura de banda da rede deve ser grande para suportar as frequentes chamadas e respostas entre clientes e servidor, impedindo a escalabilidade do modelo; a manutenção e a atualização dos clientes torna-se difícil, pois cada cliente deve ser atualizado e mantido separadamente; esta arquitetura está fortemente orientada para o acesso de aplicações a bancos de dados, não podendo usufruir da reutilização da arquitetura orientada a componentes[10].

2.3.2 – Modelo CORBA

O padrão CORBA (Commom Object Request Broker Architecture) [11] constitui-se uma tentativa de cooperação da indústria de computação coordenada pelo OMG (Object Management Group) no sentido de desenvolver um padrão aberto para possibilitar computação distribuída entre um grande número de ambientes de aplicações heterogêneos. O OMG é uma organização sem fins lucrativos, baseado em um consórcio e tem a responsabilidade de produzir e manter o conjunto de especificações e procedimentos padrão para sistemas orientados a objetos distribuídos e interoperáveis[10].

A inovação de CORBA em relação ao modelo cliente-servidor é sua orientação voltada ao modelo de orientação a objeto e sua independência em relação a protocolo, sistema operacional, linguagem de programação e plataforma de hardware[10]. A adoção de CORBA exige apenas que a aplicação, escrita em qualquer linguagem, instalada em qualquer lugar da rede, mapeie sua interface para IDL (Interface Definition Language) que trata-se da linguagem neutra do padrão, desenhada para a disponibilização e acesso a serviços (métodos e funções) de objetos remotos CORBA.

O padrão CORBA também define uma coleção de serviços a nível de sistema para a manipulação de aplicações de baixo nível na disponibilização de serviços como ciclo de vida, persistência, transação, nomeação e segurança[10]. Essas definições completam o padrão definindo aspectos principais para a interoperabilidade entre ambientes. Na versão CORBA 1.1 o foco foi direcionado para a criação de componentes com portabilidade entre aplicações orientadas a objetos, sem considerar-se o aspecto da interoperabilidade. Com a atualização do padrão CORBA 2.0 a interoperabilidade foi adicionada para que funcionasse entre diferentes vendedores de ORB (Object Request Broker), com a implementação de um protocolo chamado IIOP (Internet Inter-ORB Protocol). Esse protocolo define o que vem a ser o meio de comunicação comum (backbone) para ORB, através do qual outros ORBs podem fazer a ponte e prover acesso e interoperabilidade com os serviços associados a eles[10].

O comportamento de uma solução baseada em CORBA é baseado em um ORB que é um objeto funcionando como um mecanismo transparente, que recebe solicitações e manda respostas aos objetos pela rede, independente de qual ambiente estes objetos estejam. O ORB intercepta uma solicitação de um objeto para outro, é responsável por localizar o objeto solicitado e envia a solicitação ao destino, aguarda a resposta e entrega a resposta ao solicitante. Como parte da implementação, o ORB possibilita a interface para serviços disponibilizados em CORBA, o que permite a construção de ambientes personalizados de aplicações distribuídas[10]. Na Figura 2.3.2-4, temos uma representação

simplificada de um exemplo de arquitetura CORBA como catalisador de recursos entre aplicações escritas em C, C++ e Java interoperando através da IDL.

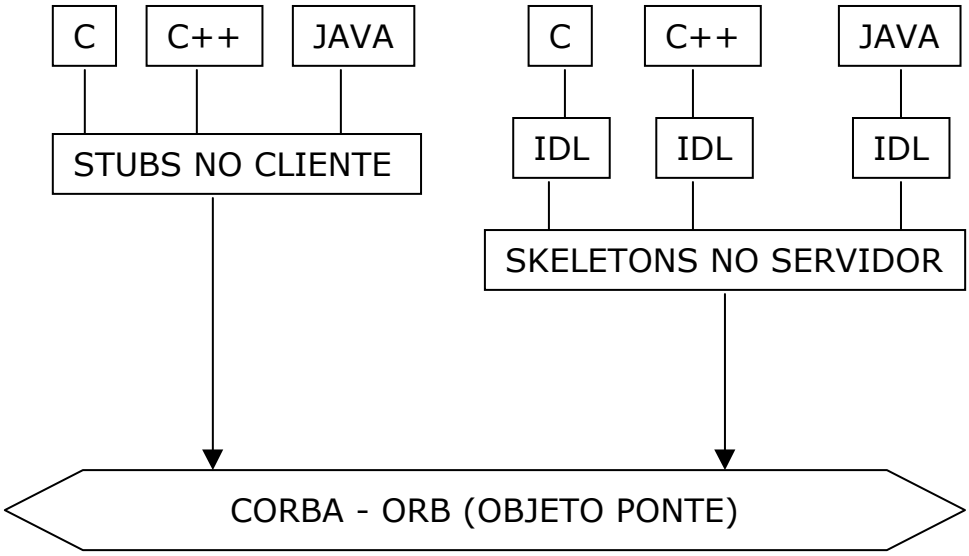


Fig. 2.3.2-4 modelo de aplicação CORBA

A arquitetura CORBA possui duas partes de destaque : IDL e ORB. IDL é empregada para estabelecer contratos, limitando a interação das aplicações e estabelecendo as interfaces com os clientes. Constituindo-se uma linguagem neutra e poderosa IDL, dispões de mecanismos para que as interfaces dos componentes de aplicações distribuídas possam especificar herança de classes, eventos, atributos e exceções em um padrão compatível com a base ORB CORBA. O ORB funciona como um objeto corredor, uma ponte, disponibilizando a infra-estrutura de comunicação para enviar e receber solicitações/respostas dos clientes para os servidores. Esse comportamento estabelece a alicerce para as aplicações orientadas a objetos distribuídas, propiciando a interoperabilidade entre ambientes heterogêneos[10].

Em comparação com o modelo de aplicação cliente-servidor podemos citar as seguintes vantagens de CORBA :

- independência de sistema operacional e linguagem de programação – a interface entre clientes e servidores é definida em OMG IDL o que possibilita as seguintes vantagens no ambiente de Internet : ambiente de aplicação em multi-lingugem e multi-plataforma, o que possibilita a separação lógica entre interface e implementação;
- integração de aplicações legadas e personalização de integração de aplicações – com o emprego de CORBA desenvolvedores podem encapsular aplicações legadas particulares e disponibilizá-las como objetos que clientes podem invocar através do ORB;

- completa infra-estrutura para objetos distribuídos – CORBA define conjunto extenso de serviços necessários a computação de objetos distribuídos como ciclo de vida, eventos, nomeação, transações e segurança;
- transparência de localização – o padrão CORBA torna a localização física dos objetos requisitados transparente. A referencia de um objeto é independente da sua localização física e do nível de localização da aplicação. Isto permite o desenvolvimento de sistemas baseado em CORBA nos quais objetos podem ser movidos entre locais físicos, sem que seja necessários alterações nas aplicações que os utilizam;
- transparência de rede – pelo uso do protocolo IIOP, um ORB pode ser interconectar com outro qualquer ORB localizado em qualquer ponto da rede;
- suporte para avisos de retorno – CORBA pode manipular eventos e permite que objetos recebam avisos e notificações de eventos assíncronos de qualquer outros objetos;
- chamada dinâmica de interface – os clientes em CORBA podem invocar objetos remotos de dois modos : estaticamente e dinamicamente. Estaticamente os métodos e objetos a serem usados devem ser definidos previamente no tempo de compilação e deve-se gerar um stub para acesso em tempo de runtime. No acesso dinâmico as aplicações tem a oportunidade de descobrir os objetos e os métodos que desejam usar, direto em tempo de runtime[10] (o emprego desta opção é muito raramente utilizado).

Uma solução baseada em CORBA deve considerar três aspectos fundamentais, que podem ser citados como responsáveis pela não disseminação em massa do padrão CORBA : investimento inicial elevado, baixa disponibilidade de serviços CORBA e baixa escalabilidade. O investimento inicial elevado ocorre pela necessidade de se implantar e treinar toda a equipe de desenvolvimento na nova tecnologia para a disponibilização da nova arquitetura, mesmo para aplicações pequenas e projetos piloto. Devido ao alto investimento inicial, o padrão CORBA especificado pela OMG continua bastante fraco e possui poucos produtos que o implementam, isso torna a quantidade de serviços distribuídos CORBA muito reduzida, quase insignificante. E devido a alta e rigorosa natureza do acoplamento orientado a conexão da arquitetura CORBA, volumes de acesso muito elevados, como os demandados por aplicações corporativas, podem fazer com que as respostas se tornem lentas, não alcançando a escalabilidade adequada[10].

Tais limitações estão sendo superadas e hoje temos a aplicação da arquitetura CORBA com IIOP no ambiente de Internet, Extranet e Intranet, em algumas iniciativas que consideram este padrão adequado e uma escolha viável para seus propósitos[10].

2.3.3 – Modelo Java RMI

Para possibilitar que aplicações desenvolvidas em Java pudessem dispor de objetos distribuídos, a Sun Microsystems desenvolveu o RMI (Remote Method Invocation) que permite a aplicações Java chamarem remotamente objetos e passarem argumentos a eles e receberem valores de retorno. Para isso emprega-se o mecanismo de serialização de objetos Java que constitui-se em uma técnica pouco complexa e leve, a qual permite a conversão de objetos em cadeias de bits que podem ser transportados pela rede e remontados no destino (streams)[10].

A solução RMI emprega o JRMP (Java Remote Method Protocol) como protocolo de comunicação interprocesso, permitindo que objetos Java residentes em diferentes máquinas virtuais Java (VM), invoquem de modo transparente os métodos uns dos outros[10]. Como as VMs podem estar funcionando em diferentes computadores da rede, temos então um cenário de computação distribuída possibilitado pelo RMI, como demonstra [22].

O sistema empregado pelo RMI para manipular objetos utilizados de modo distribuído na rede é o de contagem de referência para o mecanismo de coleta de lixo (garbage collection) que mapeia as referências dos objetos externos para os objetos locais em conexções vivas usando a máquina vitrual. Quando um objeto local é encontrado sem uma referência externa, o mesmo é considerado fracamente referenciado e será coletado como lixo[10].

Em uma aplicação que utiliza a arquitetura RMI, um mecanismo orientado a registro (rmiregistry) disponibiliza um simples serviço não persistente de procura por nome que é usado para armazenar, no cliente, a referência do objeto remoto e permitir que o mesmo seja encontrado a partir da aplicação cliente. A infra-estrutura RMI baseada em JRMP comporta-se como meio de comunicação entre os clientes RMI e objeto remoto. Ela intercepta as solicitações do cliente, passa os argumentos de invocação, delega as solicitações de invocação para o skeleton RMI e no retorno passa para o stub, no lado cliente, os valores de retorno do método executado[10]. Também são permitidas mensagens de aviso do servidor para os clientes, porém deve-se ter estrutura de tratamento de notificações no lado cliente. Na Figura 2.3.3-5 apresentamos, de modo simplificado, modelo de arquitetura de uma aplicação baseada em Java RMI.

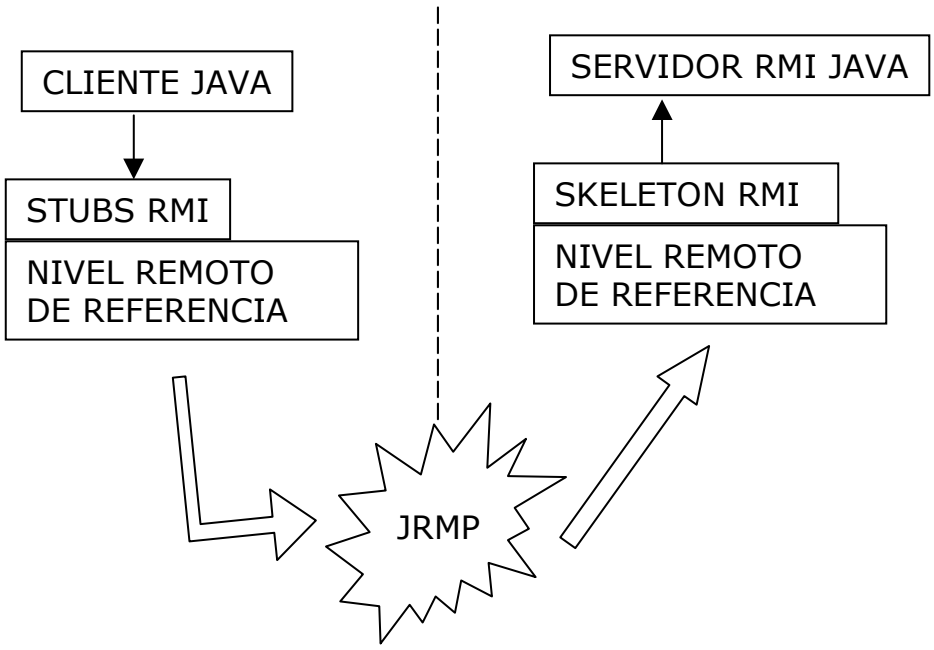


Fig. 2.3.3-5 modelo de aplicação Java RMI

Os principais componentes da arquitetura Java RMI que podemos citar são :

- cliente RMI – o cliente RMI pode ser desde um applet Java ou uma aplicação padrão Java rodando em um computador que execute a chamada a um método remoto em um servidor de objeto. Podem ser passado argumentos como tipos primitivos de dados ou objetos que possam ser serializados;

- stub RMI – constitui-se em um proxy que deve estar presente no cliente. É gerado pelo compilador rmic (que está incluído no Java developer kit-JDK) e encapsula as informações de rede sobre como acessar o servidor, delegando a invocação de métodos para o mesmo. Tem a função de empacotar os argumentos passados na chamada dos métodos e desempacotar os valores retornados pela execução dos métodos;

- infra-estrutura RMI – esta infra-estrutura é constituída de dois níveis : o nível de referência remota e o nível de transporte. O nível de referência remota separa especificamente o comportamento de referenciamento do Stub que atende ao cliente. Este nível manipula aspectos semânticos de referência (localização) como tentativas de conexão que podem ser unicast/multicast geradas pelas solicitações de invocação remota. No nível de transporte temos a disponibilização da infra-estrutura de rede a qual facilita a transferência de dados durante a chamada dos métodos na passagem dos argumentos e no retorno dos resultados da execução;

- skeleton RMI – também é gerado pelo compilador rmic, recebe a invocação remota da solicitação do cliente que o stub passa, processa os argumentos (desempacota) e delega a chamada ao servidor

de RMI. Após a execução com sucesso do método pelo servidor, empacota os resultados e devolve de volta ao stub presente no cliente através da infra-estrutura RMI.

- servidor de RMI – este servidor constitui-se em um objeto Java que implementa os métodos expostos nas interfaces e executa as solicitações dos clientes. Recebe as solicitações remotas chegadas e passadas pelo skeleton que previamente já desempacotou os parâmetros. O servidor executa o método solicitado e passa a resposta para o skeleton o qual, via infra estrutura RMI, passa ao stub o qual devolve o resultado ao cliente[10].

O desenvolvimento de aplicações distribuídas utilizando RMI é simples e natural, totalmente aderente ao modo de programação Java, toda a parte de protocolo e demais aspectos de interação entre objetos distribuídos já está definida e pronta para usar. RMI é construído sobre sockets TCP/IP, com a vantagem adicional de disponibilizar abordagem orientada a objetos para a comunicação entre processos. RMI traz ainda gerenciamento distribuído de recursos, otimização do poder de processamento e distribuição balanceada de carga, características inerentes ao modelo de programação Java[10].

Para a conexão e interoperabilidade com componentes produzidos e disponibilizado pela arquitetura CORBA foi desenvolvido o RMI-IIOP (RMI sobre o protocolo IIOP). Entretanto podemos citar três limitações consideradas importantes inerentes a RMI : RMI limita-se somente a plataforma Java.

Aplicações Java RMI são fortemente acopladas e RMI não possui mecanismo específico para gerenciamento de sessão. O fato de estar disponível apenas para a plataforma Java torna RMI dependente desta linguagem, contrariando o objetivo de independência total da computação distribuída, que é um dos objetivos do CORBA.

No aspecto relativo ao forte acoplamento das aplicações RMI isto ocorre devido a sua natureza orientada a conexão o que impossibilita alcance de alta escalabilidade, um dos requisitos principais para modelos aplicações distribuídas, em [23] encontramos exemplo acadêmico que reflete essa característica e aderência de Java a vários bancos de dados.

O fato de não dispor de mecanismo para o gerenciamento de sessão faz com que a aplicação de RMI seja limitada a alguns domínios de aplicação, pois em uma aplicação no modelo cliente-servidor, o servidor deve manter gerenciamento de sessão e o estado de informação relativo a vários clientes simultâneos que o estão acessando. A implementação do gerenciamento de sessão e do

estado de informação sem que haja um suporte natural da arquitetura adotada (no caso em RMI) constitui-se em uma tarefa complexa[10].

2.3.4 – Modelo Microsoft DCOM

Para a comunicação entre aplicações Windows baseadas em componentes, por meio de binário e em redes do padrão sistema operacional Windows, a Microsoft definiu o COM (Component Object Model) que utilizava outro mecanismo proprietário denominado OLE (Object Linking and Embedding) o qual empregava registros de objetos baseados na tecnologia Windows para a organização e gerenciamento de componentes da família ActiveX que podiam ser acessados por aplicações distribuídas[10]. Entretanto, esta solução demonstrou grande instabilidade e baixa confiabilidade.

A tecnologia da Microsoft para a computação distribuída na plataforma Windows denomina-se DCOM (Distribuited Common Object Model) que implementa mecanismo RPC através do qual aplicações COM podem comunicar-se empregando o DCOM como protocolo. Empregando a abordagem de stub e skeleton o DCOM, expõe através de interface definida os métodos de objetos COM que podem ser invocados remotamente através da rede. O cliente pode invocar métodos de objetos COM que estão situados em computadores remotos como se estes estivessem presentes localmente. O stub encapsula a localização na rede do servidor onde encontra-se o objeto COM e comporta-se como um proxy no lado cliente. Os servidores podem conter ou hospedar vários objetos COM que devem registrar-se no servidor, tornando-se assim visíveis e disponíveis para todos os clientes quem devem descobrir tais objetos através de mecanismos de busca[10].

Para a plataforma Windows, DCOM tem obtido aprovação e aceitação, porém limitados somente a plataforma Microsoft de aplicações. Entre as principais limitações dessa tecnologia podemos citar : plataforma proprietária e pertencente a um único fornecedor; não há gerenciamento de estado da informação distribuída; baixa escalabilidade e alta complexidade no gerenciamento de sessões[10]. A Figura 2.3.4-6 a seguir, demonstra o esquema simplificado da arquitetura DCOM que pode ser utilizada somente no ambiente de desenvolvimento e sistema operacional Microsoft Windows.

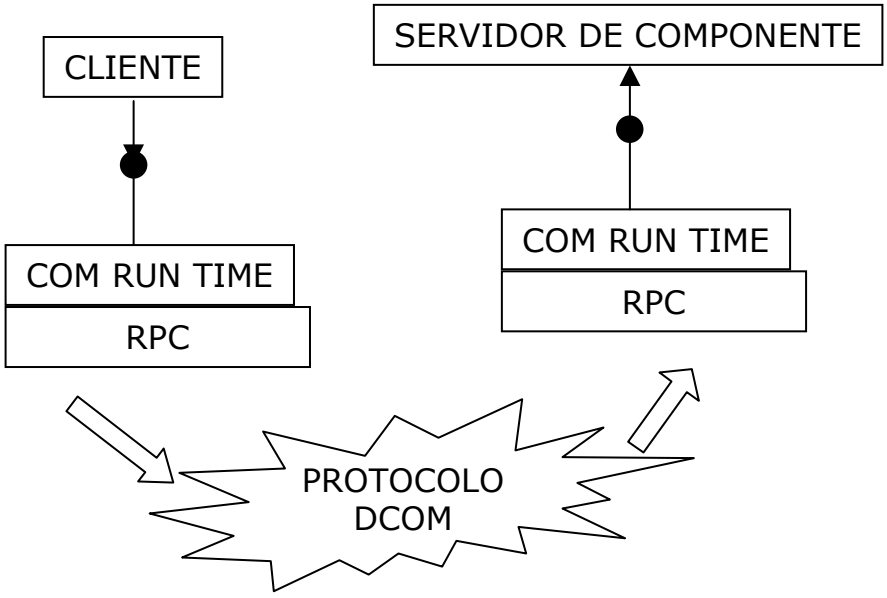


Fig. 2.3.4-6 modelo de aplicação Microsoft DCOM.

2.3.5 – Modelo Message-Oriented Middleware

As arquiteturas CORBA, RMI e DCOM possuem diferenças em suas concepções básicas de implementação. Porém todas adotam alto grau de acoplamento, adoção de mecanismos de comunicação síncrona, baseiam-se em protocolos de comunicação binários e adotam alto grau de integração entre suas camadas lógicas, todas estas escolhas de implementação fazem com que a escalabilidade, um dos desafios principais das aplicações distribuídas, seja amplamente comprometida e dificilmente alcançada[10].

O modelo de MOM (Message-Oriented Middleware) é baseado em baixo acoplamento, comunicação assíncrona no qual o cliente não necessita saber nada sobre os recipientes da aplicação servidora ou quais métodos devem ser chamados. O modelo MOM possibilita a comunicação indireta entre aplicações disponibilizando uma fila de mensagens. A aplicação cliente envia mensagens para a fila de mensagens (uma área que armazena as mensagens) e a aplicação destino deve ter a iniciativa de retirar da fila as mensagens destinadas a ela. Nesse modelo, a operação que envia a mensagem continua a funcionar após o envio, sem ter que ficar aguardando a resposta da aplicação destino[10].

Na arquitetura MOM uma aplicação interage com a infra-estrutura de mensagens através de adaptadores personalizados para cada ambiente de aplicação. Clientes e servidores podem enviar e retirar mensagens da infra-estrutura através destes adaptadores. Caso haja necessidade de se aumentar a confiabilidade das mensagens entregues, pode-se persistir as mesmas em um banco de dados ou sistema de arquivos[10]. Existem várias soluções proprietárias para este modelo de aplicação distribuída, uma

das mais interessantes é o JMS (Java Message Service) que possibilita comunicação ponto a ponto, sistema de mensagens baseado em publicão/inscrição, capacidade de gerenciar transações, confiabilidade na entrega de mensagens e segurança.

A implementação de sistemas de aplicações distribuídas baseados em MOM apresenta alguns desafios, entre os quais podemos citar dois : a maioria dos padrões de MOM implementados possuem APIs nativas usada para a comunicação com o núcleo de sua infra-estrutura o que afeta a portabilidade das aplicações através de diferentes implementações e pode prende-la a um fornecedor específico. E o outro desafio é que as mensagens utilizadas para integrar aplicações no modelo MOM são geralmente baseadas em formatos proprietários sem que haja qualquer tipo de padronização sobre elas, o que torna a compatibilidade com outros sistemas uma questão complexa[10]. A Figura 2.3.5-7, apresenta uma representação simplificada da arquitetura MOM.

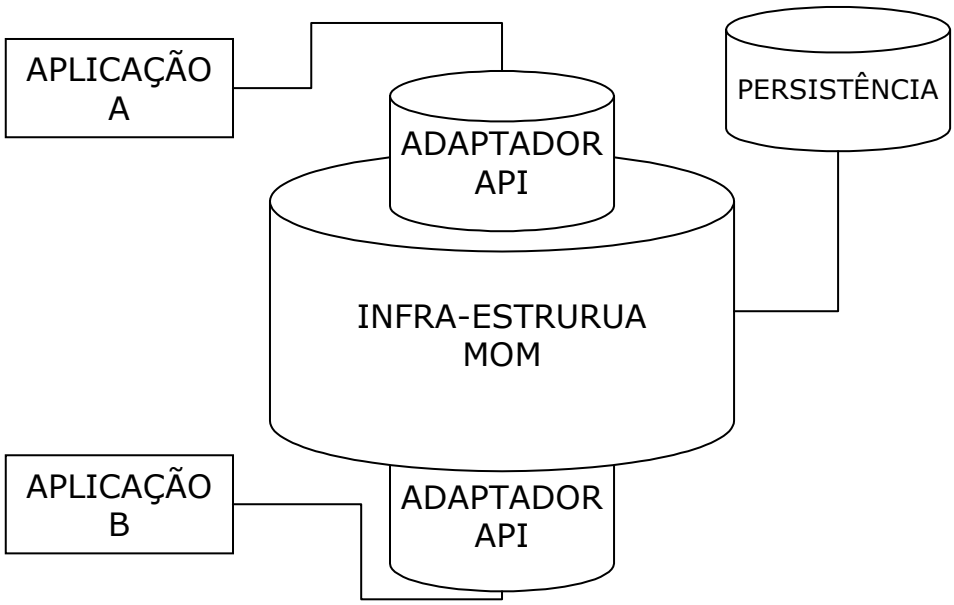


Fig. 2.3.5-7 modelo de aplicação MOM (Message-Oriented Midleware).

2.3.6 - Desafios da computação distribuída no ambiente da Internet

Os modelos de computação distribuída discutidos até aqui como CORBA, RMI e DCOM obtiveram relativo sucesso na integração de aplicações em ambientes homogêneos dentro de redes locais. Porém a Internet expandiu-se para conectar negócios, o que demanda interoperabilidade entre as aplicações através da rede[10] criando assim um novo cenário com especificações e exigências (desafios) adversas para os modelos citados anteriormente.

Podemos citar como desafio a dificuldade de alta complexidade para a manutenção de versões atualizadas de componentes skeleton e stubs nos clientes e servidores em ambientes de rede heterogêneos. Características como velocidade de resposta, disponibilidade e a capacidade de processar grande volume de dados em ambientes distribuídos ainda consomem grande parte do tempo de desenvolvimento destas aplicações. A interoperabilidade entre aplicações ainda é tímida e as vezes impossível entre aplicações que funcionam em redes com protocolos e plataformas heterogêneas. Temos ainda problemas de projetos pois alguns dos protocolos adotados não foram concebidos para trafegar através de firewalls ou para ser acessados via padrões de Internet[10].

2.3.7 - J2EE e XML como modelos alternativos para a computação distribuída

A Internet e seus padrões impuseram um modelo de programação no qual o cliente deve ser o navegador web (browser) e o servidor deve resolver toda a complexidade da aplicação e responder o solicitado no mais simples HTML possível. Para atener a esta especificação J2EE (Java 2 Enterprise Edition) disponibiliza modelo de programação baseado na Web e em componentes de negócios gerenciados por um servidor de aplicação J2EE[10]. Esse servidor de aplicação constitui-se de um conjunto de API e serviços de nível básico disponíveis para os componentes. Como serviços de nível básico podemos citar : segurança, suporte a transação, conexões, gerenciamento de pooling, serviços de concorrência. Todas essas facilidades devem favorecer ao desenvolvedor que pode focar exclusivamente sobre a lógica de negócio a ser implementada[10]. Esse modelo está baseado em Java e em padrões e especificações da indústria disponibilizando interfaces para a conexão com vários sistemas legados e sistemas de informação (ERP). Possui capacidade de conexão com um conjunto diversificado de clientes que variam de PDA (Personal Digital Assistant) até o navegador web (browser), permite ainda conexões com clientes considerados "ricos" como applets Java, CORBA e aplicações Java rodando em computadores pessoas ligados em rede[10].

J2EE possui arquitetura típica fisicamente dividida em três camadas lógicas, as quais separam claramente os vários componentes da aplicação de modo a atribuir papéis e responsabilidades para cada parte. A primeira camada é a camada de apresentação que é composta pelos componentes Web que gerencia e manipula solicitações e respostas HTTP, gerenciamento de sessão, entrega de conteúdo independente de cliente e a chamada de componentes de negócio[10]. Esta camada interage diretamente com o cliente, seja de que tipo for, browser, PDA e outros.

Temos a seguir a segunda camada, chamada de camada de aplicação (também denominada camada de negócios) a qual contém o núcleo de toda a lógica de processamento, o que

tipicamente chamamos de fluxo de dados e automação. Os componentes de negócios desta camada recuperam informações de fontes de dados, que podem ser bancos de dados ou sistemas de informação (ERP), através de API bem definidas disponibilizadas pelo servidor de aplicação.

A terceira camada de integração tem a função de comunicar-se com os sistemas de retaguarda que podem ser ERPs, EIS (Enterprise Information Systems), bancos de dados, sistemas legados ou aplicações que funcionam em mainframes[10].

O particionamento da aplicação em camadas propiciou a adoção de J2EE como padrão para aplicações críticas baseadas no ambiente de Internet. O padrão J2EE pode possibilitar a interoperabilidade entre aplicações J2EE, permitindo que componentes de negócio colaborem entre si através da rede sem que seja necessário a interação humana. Também é possível a sindicalização e colaboração entre processos de negócios que compartilhem dados e componentes no nível de processamento em tempo real através da rede. A este tipo de fenômeno chama-se geralmente de comunicação negócio-a-negócio (B2B business-to-business)[10].

O surgimento de XML (Extensible Markup Language) como meio capaz de transportar dados estruturados, agregando a esta estrutura sua própria descrição e formato, fez com que o mesmo fosse logo adotado como padrão pela industria para a troca eletrônica de dados e meio de comunicação.

Desta forma o emprego de XML para a troca de dados entre aplicações contribui para aumentar a interoperabilidade entre elas e acrescenta escalabilidade para as aplicações de retaguarda. A combinação de XML e do padrão J2EE oferece um alternativa completa para a interoperabilidade entre aplicações de negócios no que se chama de B2B[10].

Assim, temos J2EE como uma possibilidade de padrão para a construção de aplicações na Internet e XML para que estas aplicações transmitam/recebam informações entre si, funcionando como "cola" no que diz respeito a interoperabilidade. A Figura 2.3.7-8, apresenta esquema simplificado da arquitetura J2EE, na qual pode-se observar a presença destacada das três camadas, do emprego do conceito de containers e da interoperabilidade que o modelo J2EE possibilita entre diversificado conjunto de clientes, aplicações legadas e fontes da dados.

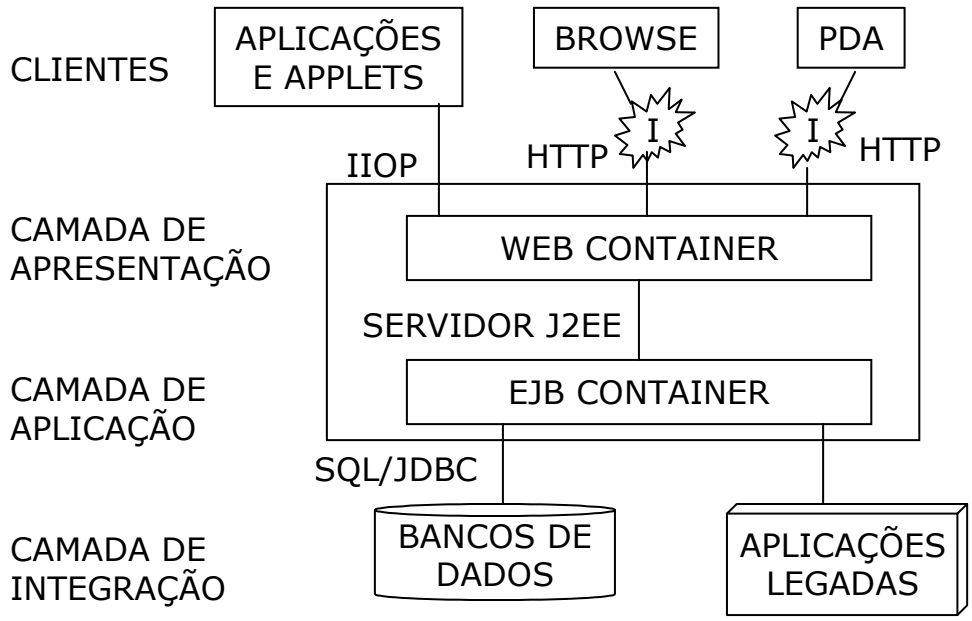


Fig. 2.3.7-8 arquitetura de aplicação J2EE.

Na arquitetura J2EE destaca-se a divisão em três camadas, capaz de atender a grande diversidade de clientes. Tomando como base a Figura 2.3.7-8, passaremos a discorrer a função de cada uma das camadas representadas, no sentido de cima para baixo. A Camada de Apresentação aloja a parte de interação com o cliente. Nessa camada estão as páginas HTML, JSP (JavaServer Pages), os Servlets, imagens e demais artefatos que devem ser transferidos para os clientes. Esta camada é responsável por emitir respostas e extrair da camada de aplicação as informações necessárias ao atendimento das solicitações de usuários [57] .

A lógica da aplicação, as restrições, as regras de negócio e a modelagem semântica de componentes das aplicações em J2EE, são armazenadas na Camada de Aplicação que hospeda os EJBs (Enterprise Java Beans) [6],[7] que tem a função de extrair os dados da Camada de Integração e repassá-los, agregando valor semântico e processamento, para a Camada de Apresentação. Os EJBs situados nesta camada devem ser construídos dentro da metodologia de componentes [17] prevendo-se a sua reutilização em várias aplicações [39]. O emprego de padrões de projeto (design patterns) nesta camada representa comportamento fundamental para a obtenção do desempenho esperado para as aplicações que utilizarão estes componentes como alerta [38].

Para a arquitetura J2EE, a Camada de Integração é o lar das fontes de dados que podem ser : arquivos comuns, bancos de dados, aplicações antigas, sensores etc.[57]. Essa camada tem a função de fornecer para a Camada de Aplicação os dados necessários para aplicação. Nas aplicações convencionais e corporativas esta camada é responsável pela persistência dos dados, mantendo estes

dados para consulta, alteração e exclusão definitiva, enquanto a Camada de Aplicação desejar, como demonstra [41], [42] e [43].

2.4 - Web Services – conceitos fundamentais

As arquiteturas tradicionais de sistemas distribuídos incorporam relativa fragilidade pois acoplam fortemente vários componentes ao sistema. Esses sistemas distribuídos por sua vez demonstram-se altamente sensíveis a mudanças. Na Web o volume de mudanças cresce rapidamente no que se refere as necessidades dos negócios de modo a colocar os sistemas distribuídos em crise e em posição de fraqueza por não poder acompanhar tais mudanças, devido ao alto grau de acoplamento e interdependência entre as partes que compõem os modelos. Qualquer mudança que ocorra em um dos componentes pode fazer com que todo o sistema venha a falhar ou tornar-se instável[3].

Diante da perspectiva de que modelos tradicionais de sistemas distribuídos demonstram certa dificuldade para acompanhar a dinâmica da Internet e da constatação de que a simplicidade de interação, característica do modelo de programação Web (discutido na Seção 2.2), possibilita a construção de sistemas de modo incremental, com adição de clientes e servidores de maneira não centralizada, sem uma coordenação central[1]. Sendo que todas estas características ocorrem naturalmente dentro dos fundamentos e das bases da Internet[4]. Surge a necessidade de se estabelecer uma forma que possa ser aderente aos padrões já existentes e permita a comunicação entre programas.

Uma solução possível para as questões de integração de aplicação-para-aplicação deve manter a natureza fundamental da Web preservando a coordenação descentralizada e a flexibilidade da integração de sistemas e modelos diferentes. Esses pontos garantem a interoperabilidade, baseada em um conjunto pequeno de padrões como HTTP e HTML que permitem certa convergência de outros ramos da pesquisa no sentido de agregar diversidade de participantes e novos serviço[4]. Outro ponto a ser preservado dentre as características fundamentais da Web é a grande simplicidade para a disponibilização de recursos, que torna a adição de clientes, servidores, aplicações e demais recursos altamente facilitada.

Com o avanço natural da Internet, as aplicações Web nas quais uma aplicação interage com outra aplicação, passam a ocupar espaço e a demandar modelos para que se garanta mais confiabilidade, disponibilidade e robustez a este uso da Web. Exemplos são conexões para o desenvolvimento automatizado de mercados e automação de transações entre parceiros de negócios.

Nesse novo uso da Internet devemos considerar dois aspectos : qual o modelo de aplicação distribuída a ser utilizado para aplicações baseadas em Web de negócio-para-negócio (aplicação-para-aplicação) e como integrar plataformas antigas de sistemas distribuídos preexistentes, sendo utilizados em redes locais privadas, que agora precisam interagir com o ambiente de Internet. O modelo de Web Services é uma das respostas oferecidas a estas questões[4].

A utilização crescente de Web Services é impulsionada por vários fatores, um deles de maior relevância, é o fato de que são baseados em XML, o que facilita a integração e colaboração interna e com parceiros. Mesmo ainda apresentando alguns pontos e partes que necessitam de amadurecimento[5] essa tecnologia encontra grande volume de aplicação na área de negócios, pois representa verdadeira revolução no relacionamento entre parceiros, oferecendo a próxima geração de integração entre empresas de forma mais fácil e mais barata. Assim, reúne em uma única tecnologia facilidade, rapidez que podem ser, mesmo de forma tímida, implementadas hoje.

Outra forma de descrever a aderência de Web Services ao ambiente de computação distribuída Web, é através de três características de concepção básica desta tecnologia : Web Services empregam SOAP[1] para a troca de mensagens, contém metadados que descrevem o serviço disponibilizado[1] e possuem diretório integrado para descrição e localização de Web Services[14]. Estas três características são ordenadas através de uma arquitetura própria orientada a serviços.

2.4.1 - Protocolo para troca de mensagens

Tomando como base a XML, definiu-se o protocolo puramente baseado nesta linguagem denominado SOAP de Single Object Acess Protocol, que tem a finalidade de tornar-se uma lingua franca[4] para o Web Services, ou seja este protocolo será usado para toda e qualquer comunicação e utilização de Web Services. O protocolo SOAP será o protocolo de mensagem para transporte de conteúdos entre Web Services e seus programas clientes.

O protocolo SOAP foi projetado para ser independente de plataforma (devido a utilização da XML), as mensagens SOAP deverão transportadas sobre qualquer protocolo arbitrário[4]. Como protocolo ainda em desenvolvimento, o SOAP requer algumas definições e padronizações.

Para o ambiente Web, mensagens SOAP representam uma mudança de tipo MIME (Multipurpose Internet Mail Extensions)[4] para mensagem baseadas em XML. Clientes deste ambiente são geralmente navegadores (browser) que possuem a capacidade de transformar HTML e alguns formatos de MIME em dados que são apresentadas no navegador web (browser), deixando a interpretação semântica da informação para o usuário[1].

Mesmo estando em XML o conteúdo de uma mensagem SOAP, não carrega consigo mesmo seu significado semântico completo. Por isso o projetista do Web Services deve decidir que informações cada mensagem deve carregar e qual a relevância desta para o Web Services a ser oferecido como um todo[1]. Em alguns casos podemos ter que utilizar mensagens enviadas contendo parâmetros para que outros métodos possam ser chamados e utilizados na seqüência, dentro do mesmo uso do Web Services corrente.

Uma das vantagens deste tipo de abordagem para esta tecnologia é a similaridade guardada com o modelo RPC. Entretanto precisamos mencionar que esta vantagem pode causar a restrição de se criar um certo grau de acoplamento entre fornecedores de Web Services (servidores) e clientes no que se refere as expectativas de que certas mensagens conterão certos tipos de dados (forma e conteúdo). Isto ocorre principalmente, quando da construção de sistemas centrados em métodos, que operam com a possibilidade de receber mensagens compostas por determinada quantidade de parâmetros, em ordem preestabelecida e com tipos definidos para cada um dos mesmos[1].

Em alguns casos, este tipo de abordagem faz com que se renuncie ao baixo acoplamento, muito mais pelo emprego equivocado da tecnologia apresentada do que por qualquer outro fator. Entretanto isto pode ser evitado quando por concepção, clientes e Web Services consideram em seus processos de construção as características do ambiente onde irão operar e conferem alto grau de importância ao ingrediente flexibilidade principalmente no que diz respeito ao projeto e implementação das mensagens que serão trocadas[1]. O modelo de programação Web tem seu foco muito mais voltado para o envio de uma mensagem e não como a mesma deva ser processada, está é uma premissa a ser considerada na flexibilização de sistemas para este ambiente.

O protocolo SOAP foi concebido como uma evolução dos protocolos de mensagem da Internet e traz em seu projeto dispositivos como os Cabeçalhos (headers) para facilitar sua transmissão através de vários protocolos de transporte que possuam a capacidade de interpretar este dispositivo de

[4] Segundo [13] , MIME constitui-se em padrão para o processamento de e-mail, desenvolvido pelo IETF(Internet Engineering Task Force), com o objetivo de disponibilizar suporte para mensagens compostas de partes múltiplas e/ou multimídia.

otimização, oferecendo vários serviços de alto nível como por exemplo segurança, garantia de qualidade de serviço, banda e rotas privilegiadas[1]. O emprego deste recurso abre a possibilidade para a virtualização da topologia da rede utilizada pelos Web Services, suas mensagens SOAP e seus clientes de tal modo que será possível decidir que caminho e que tipos de protocolos um Web Services poderá utilizar para atender a um cliente[1].

Outros protocolos podem ser usados para o transporte de envelopes SOAP como por exemplo SMTP, raw TCP, FTP [1].

2.4.2 - Metadados para descrição funcional abstrata

A maior das três diferenças entre Web Services e outras tecnologias para a implementação de sistemas distribuídos na Internet é a de que existe, por definição, uma auto descrição pública dos Web Services. São publicadas especificações empregando metadados, que descrevem mensagens produzidas e consumidas, padrão de troca de mensagens usados para expor comportamentos, protocolos de transporte físico usado, endereço lógico para acesso ao mesmo. Os formatos das mensagens usadas em Web Services são definidos usando XML Schema (XSD) [1].

Um Web Service é descrito usando WSDL (Web Service Description Language) [11] que mapeia mensagens trocadas para operações agrupadas por Types (tipos, que na realidade são interfaces) e descreve como estas operações podem ser chamadas usando um protocolo particular de transporte aderente[1]. Esta descrição pode ainda ser usada para se comunicar com o Web Service direta ou indiretamente.

A neutralidade do XML, garante independência de plataforma, de tal modo que fornecedor e consumidor de Web Services podem selecionar o protocolo que melhor lhe convier[4], desde que este esteja descrito na WSDL. Neste modelo, o suporte a heterogeneidade é fornecido pela descrição funcional abstrata do Web Service que deve estar separada, ao máximo, dos limites dos protocolos disponibilizados para acesso[4].

Por meio desta descrição ocorre uma representação uniforme da aplicação, do serviço a ser acessado, independente do protocolo ou plataforma a ser utilizada. Esta descrição abstrata informa detalhes necessários ao uso do Web Services[4].

O WSDL funciona como o catalisador entre o fornecedor do serviço e o consumidor. Ambos, tem acesso ao WSDL e podem comunicar-se trocando informações em um formato previamente descrito, não importando em que linguagem, sistema operacional ou plataforma os participantes da conversação estejam. Nisto se constitui a principal função do WSDL[15].

2.4.3 - Diretório integrado para a descrição, publicação, procura de serviços na web

O terceiro elemento que compõe a tecnologia de Web Services provê método integrado para descrição, publicação, descoberta (procura) de informações sobre os serviços oferecidos na Web[14]. Trata-se de um serviço de diretório em forma de iniciativa da indústria que estabeleceu um modo aberto, padronizado e independente de plataforma focado no processo de facilitação da localização de Web Services disponíveis.

Este componente denominado UDDI de Universal Description, Discovery, and Integration constitui-se em uma iniciativa pública, composta por dois grupos : o grupo de trabalho e o grupo consultivo. O grupo de trabalho é responsável por desenvolver as especificações e o grupo consultivo tem a tarefa de provê requisitos e revisar especificações. Para fazer parte do grupo de trabalho é necessário ser convidado pelos membros do grupo. O grupo consultivo porém é aberto a todos[14].

A necessidade do UDDI, o registro ocorre quando o número de Web Services a serem considerados para a escolha e utilização cresce muito. Nesse cenário, como garantir que todos os potenciais parceiros que oferecem Web Services aderentes aos requisitos procurados foram realmente considerados na escolha. Este modelo possibilita a interação formal com um grande número de possíveis parceiros e todas as interfaces por eles expostas. Conceitualmente, UDDI é um registro de Web Services nos vários nós que oferecem este serviço[14].

Antes do aparecimento do projeto UDDI, não havia na indústria forma padronizada de oferecer a clientes e parceiros maiores informações sobre produtores e Web Services disponibilizados. Não havia método uniforme e padrão que detalhasse como integrar os sistemas e os processos já estabelecidos e disponíveis e também como esta integração se daria entre parceiros de negócios que viessem a se interessar pelos Web Services oferecidos[14].

Conceitualmente, um negócio pode ser exposto de três formas no registro UDDI. No caso de Web Services apenas uma das três formas é considerada funcionalmente participante da

tecnologia. Porém as especificação de UDDI fornecerá uma boa noção de como um negócio pode ser exposto neste registro : páginas brancas, páginas amarelas e páginas verdes. Nas páginas brancas estão disponíveis informações sobre contato e identificadores das empresas, incluindo nome do negócio, endereço, regras de tributação nas quais ela esta classificada e dados relativos a cadastros oficiais. A função das páginas brancas é permitir a descoberta de Web Services oferecidos a partir da identificação de cada empresa.

As páginas amarelas descrevem Web Services usando diferentes categorias. Estas páginas permitem a descoberta de Web Services, segundo um grupo, uma classificação que coloca junto todos os similares.

No caso das páginas verdes, estas estão incluídas na tecnologia de Web Services e fornecem informações técnicas que descrevem o comportamento e as funções suportadas pelo Web Service oferecido pela empresa. Estas informações incluem apontadores para grupos específicos de informações mais especializadas (em XML e de como obter o WSDL) e também indicam onde os Web Services podem ser localizados[14].

Sob a perspectiva de quem utiliza diretamente via navegador web (browser), UDDIs podem ser considerados, de modo simplificado, como motores de pesquisa da Internet focados em um tipo especial de informações, publicadas formalmente dentro de um padrão para expor negócios e Web Services[14]. Com tais características os UDDIs oferecem interfaces muito áridas e de difícil consumo para o usuário humano.

Tipicamente motores de pesquisa da Internet como o AskJeeves, Miner, Google e outros organizam e indexam URL (Unirversal Resource Loader) e disponibilizam para usuários com um formato adequado e dentro de seu próprio estilo. Entretanto no caso da oferta de Web Services, apenas a URL seria muito pouco para conquistar um cliente em potencial, neste segmento de negócio são necessários mais detalhes[14].

Na busca por Web Services que atendam aos requisitos de sua aplicação, um analista de negócios, dificilmente consulta um UDDI diretamente. Ele deve procurar em um ou mais UDDI por Web Services que melhor se adeqüem aos seus requisitos e especificações. Para tornar esta procura mais aderente ao ser humano e facilitar o trabalho de busca por Web Services, uma série de sites especializados em negócios e outros sites de mercado, oferecem o serviço de extrair informações dos UDDI, tratá-las e torná-las mais adequadas para a tomada de decisão e análise humana[14], oferecendo facilidades que permitem a comparação entre Web Services similares disponíveis em UDDIs distintos.

Sob o ponto de vista dos desenvolvedores de Web Services, são disponibilizadas APIs para a interação com UDDIs. Nestas interações incluem-se a capacidade de publicação no UDDI (inclusão de descrição de Web Services) e também consultas a UDDI na busca por Web Services empregando-se vários critérios flexíveis de busca[14].

No que se refere a arquitetura do projeto UDDI, temos o arranjo denominado UDDI Business Registry (UBR de UDDI Registro de Negócios), também conhecido como Nuvem Pública que é conceitualmente um sistema simples composto por vários nos que tem seus dados replicados e sincronizados mutuamente. Na prática, temos um conjunto de nós que contém uma cópia do conteúdo de todos os outros. O agrupamento global de todos estes nós e seus conteúdos replicados constitui o que chamamos de UBR. Um operador (nó participante do UBR) replica seu conteúdo entre todos os outros. Acessando individualmente um operador obtém-se a informação e a qualidade de serviço disponível nos demais. Existe porem, uma regra básica que rege esta arquitetura : um conteúdo inserido em um operador fica sendo de propriedade deste operador e somente através dele o conteúdo disponibilizado pode ser alterado e ou excluído[14]. A Figura 2.4.3-9 demonstra como funciona esta arquitetura. Assim para incluir o registro de um Web Services em um UDDI o fornecedor do mesmo deve conectar-se ao UDDI desejado e preencher todos os atributos exigidos do mesmo. A configuração federativa do UDDI faz com que ao incluir-se um registro de Web Services em determinado UDDI, este registro seja replicado em todos os UDDIs que compõe a federação em questão. Existe a limitação de que alterações ou mesmo a exclusão do registro no UDDI do Web Services só poderá ocorrer via UDDI no qual o mesmo foi incluso. As especificações UDDI referem-se aos atributos que descrevem cada Web Services publicado e os esquemas estão relacionados ao modelo de dados [47] adotado.

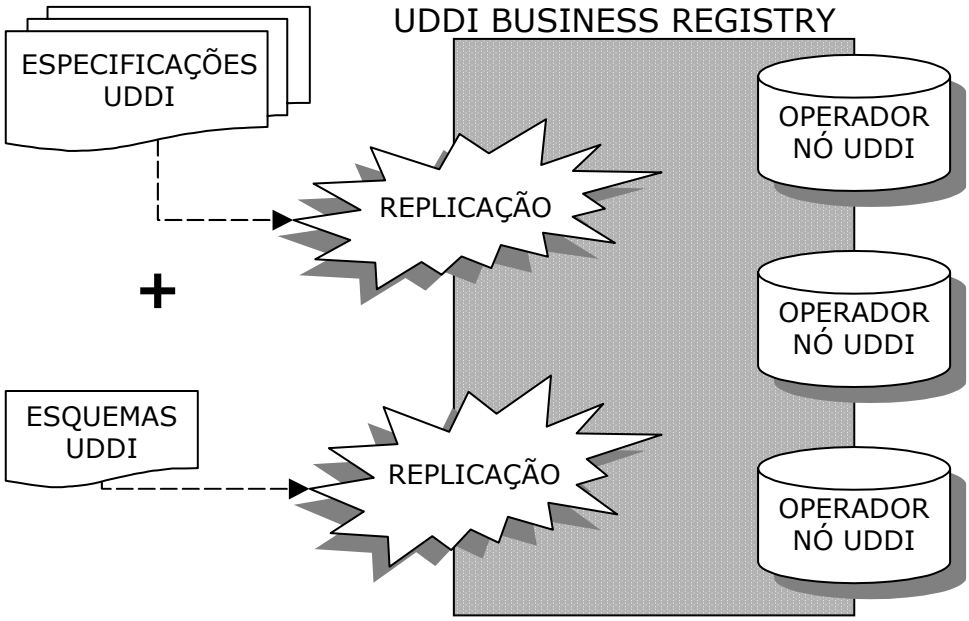


Fig. 2.4.3-9 arquitetura simplificada do UDDI.

O modelo UBR disponibiliza grande quantidade de serviços de consultas, porém restringe a publicação de serviços somente para entidades devidamente cadastradas e autenticadas por alguma entidade certificadora. Qualquer empresa ou instituição pode criar e operar um UDDI na Internet e juntar-se ao UBR. Entretanto, UDDIs privados podem definir as suas próprias regras de acesso e publicação de modo independente do padrão adotado pelo UBR.

2.4.4 - Arquitetura orientada a serviço

Um dos princípios fundamentais dos Web Services está na sua concepção baseada em SOA – Service Oriented Architecture que tem a função de coordenar o relacionamento e as principais características dos componentes de modo a oferecer suporte dinâmico a pesquisa e uso automatizado destes [3].

Em um ambiente orientado a serviços, Web Services devem ser projetados para desempenhar as seguintes atividades essenciais : possuir interfaces e métodos definidos claramente, ser publicados em uma Intranet ou em vários repositórios da Internet, para que seus potenciais usuários possam localiza-los de forma automática. Devem ainda oferecer algum benefício ou agregar algum tipo valor ao serem acionados e possuir gerenciamento de desativação quando passarem a não ser mais necessários ou sofrerem obsolescência[3].

Os componentes de uma SOA, portanto de um Web Services elaborado a partir de tal, podem desempenhar um ou mais dos seguintes papéis :

- provedor do serviço - que publica, que disponibiliza o ponto de acesso, onde se pode obter o serviço;
- corretor, o diretório do serviço – onde se publica, entidade onde se anuncia e descrevem os serviços disponíveis nos provedores, uma entidade que funciona como um catálogo onde se procura um serviço específico;
- consumidor, solicitante – entidade que procura por um serviço no Corretor, no diretório, e ao encontrar esse serviço, dirigi-se ao Provedor do serviço para consumi-lo[3].

Esses três componentes devem executar três operações básicas entre si : operação de publicação, de registro do serviço, dos métodos e da interface no Corretor. Operação de busca, de consulta do Corretor pelo Consumidor que está interessado em encontrar o serviço e finalmente, operação de consumo do serviço a partir do Consumidor que pode usufruir o serviço disponibilizado pelo Provedor[3]. A Figura 2.4.4-10 a seguir representa a arquitetura SOA, com seus componentes e as relações entre eles, os números indicam uma possível ordem de comunicação entre estes, sendo que 2, ocorre mais do 1 e 3 ocorre mais do que 2.

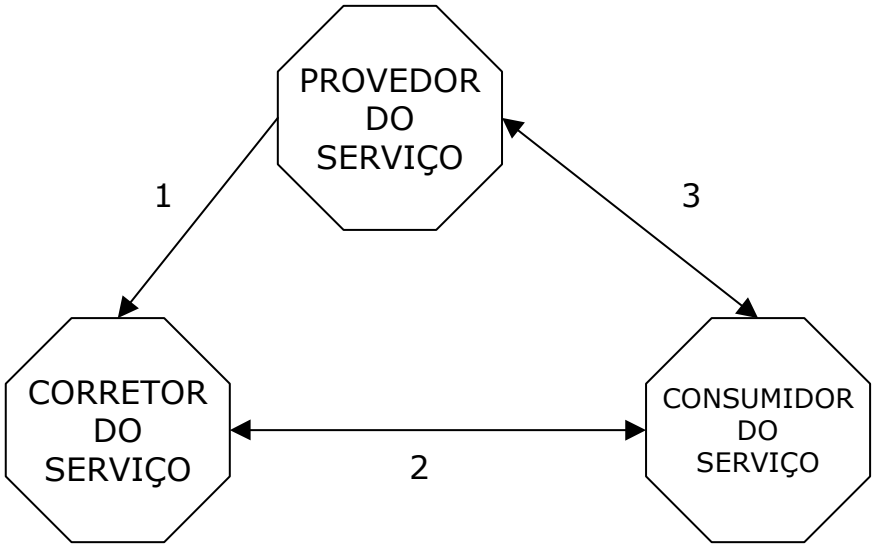


Fig. 2.4.4-10 arquitetura SOA (Service-Oriented Architecture).

2.4.5 Linguagem neutra empregada para a descrição dos componentes de Web Services

Os três principais componentes da tecnologia Web Services (SOAP, WSDL e UDDI), estão baseados sobre linguagem neutra considerada aderente aos padrões do ambiente Web. Oficialmente endossada, em fevereiro de 1998, padrão para formato de dados na Internet pelo W3C (Worldwide Web Consortium) a linguagem XML (Extensible Markup Language) passou de padrão de *facto* para padrão de direito. Empregando Unicode para codificar seus caracteres, XML é projetada de modo a conter sua auto-descrição de dados representados neutra, podendo armazenar como um simples documento dados altamente complexos de modo legível e processável. Com esta flexibilidade, XML tornou-se o padrão para o transporte de dados estruturados, conteúdos e formato para dados que representam documentos em meio eletrônico. Antes mesmo de ser declarado oficialmente como padrão pelo W3C, XML já havia se tornado língua franca, sendo universalmente aceita e adotada para a troca de informações entre aplicações, sistemas e serviços através da Internet[10].

No núcleo da tecnologia de Web Services, XML desempenha o papel vital de ser facilmente transportável, compatível e comum com a camada de rede pela qual é transmitida em todos os formatos de comunicação[10]. A Figura 2.4.5-11 demonstra graficamente como XML posiciona-se na base dos três principais padrões adotados para constituir a tecnologia de Web Services. Outros padrões que também são compatíveis com Web Services, empregam XML como sua linguagem básica.

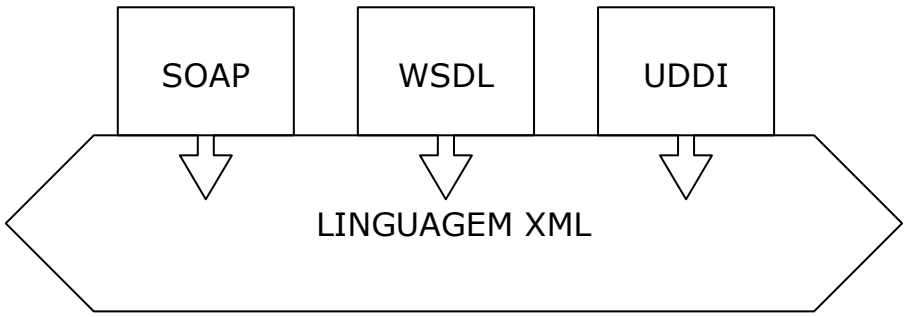


Fig. 2.4.5-11 linguagem XML como base de Web Services.

2.4.6 - Linguagem XML orientada a negócios

Para o segmento de negócios temos ainda ebXML (eletronic business XML[11]) derivada a partir do XML, que define relações globais para o mercado, no qual empresas podem estabelecer seus processos de negócio através de colaboração e transação no ambiente de Internet[10]. Também são definidos conjuntos de especificações segundo as quais empresas devem comportar-se dentro de padrões, no que diz respeito a modelagem de informações de negócios, colaboração em

processos de negócio, requisitos para o estabelecimento de parcerias, estabelecimento de acordos e troca de mensagens. Constitui-se uma iniciativa da UN/CEFACT (United Nations Center for Trade Facilitation and Eletronic Business) e outras organizações de padronização para o comercio eletrônico todas sem fins lucrativos[10].

O padrão ebXML foi adotado por vários segmentos da atividade empresarial em áreas como a saúde, o turismo e a indústria de aplicações para computadores de modo geral. A colaboração do ebXML na tecnologia de Web Services é fornecida através da definição, para a tecnologia de Web Services, de padrões para comércio eletrônico e no processo B2B, como por exemplo padrões para processos de negócio, requisitos para parcerias e acordos, registro e repositórios para Web Services, serviços de mensagens e componentes de núcleo empregados na infra-estrutura de Web Services[10]. Seu posicionamento na tecnologia de Web Services é de complemento e extensão dos outros padrões como SOAP, UDDI, WSDL, esta contribuição ocorre mais fortemente nos seguinte pontos :

- EbXML BPSS (Business Process Service Specification) define que os procedimentos e processos de negócios devem ser definidos;
- EbXML CPP/CPA (Colaborative Protocol Profile/Colaborative Protocol Agreement) especificam como os perfis e requisitos de paceiros e acordos devem ser definidos, indicando como a coreografia das transações de negócios deve ocorrer.
- EbXML MSH (Messaging Service Handler) manipula o transporte, roteamento, empacotamento de mensagens e também disponibiliza confiabilidade e segurança, agregando valor ao protocolo SOAP.

EbXML define o funcionamento dos serviços de registro, protocolo de interação e definição de mensagens para os diretórios onde os Web Services serão publicados, comportando-se como um dispositivo de armazenamento para as informações compartilhadas com os demais registros/repositórios ebXML. Os registros/repositórios ebXML comunicam-se mutuamente, como em uma federação, o que pode ser descoberto via UDDI. Esta característica permite a UDDI pesquisar toda a lista de negócios disponíveis, através de apenas um registro/repositório ebXML [10].

Os componentes núcleo do EbXML estão listados no catálogo de processos de negócios que disponibilizam as funcionalidades mais comuns para a utilização do ebXML neste segmento. Como exemplo podem ser encontrados componentes para aplicações de pagamento, inventário, cotação de preços[10].

2.5 - Arquitetura e blocos básicos de construção de Web Services

A arquitetura de Web Services é baseada em três papéis básicos e três relacionamentos. Nos papéis temos os serviços de implementação e provimento do Web Services, o serviço de corretagem e divulgação e o serviço de consumo ou solicitação. Na parte de relações ocorrem as relações binárias de registro, de procura e busca por Web Services e a relação de invocação ou uso dos mesmos.

Os princípios básicos que reúnem e ordenam elementos constitutivos Web Services é a arquitetura SOA e o ambiente de protocolos da Internet. A solução para a composição de aplicações de Web Services está baseada em padrões e em tecnologias de padrões base. O que garante que a implementação de aplicações Web Services sejam aplicações compatíveis com as especificações padrão, logo favorecendo a interoperabilidade entre aplicações compatíveis[10]. A seguir listamos os princípios chaves necessários ao projeto e implementação de Web Services :

- definir a aplicação em termos de componentes modulares para uma interface universal e um modelo de solução consistente, de tal modo que permita a exposição de serviços;
- definir padrão de procedimentos e processos baseados no modelo de infra-estrutura padrão e em protocolos que suportem aplicações baseadas em serviços no ambiente de Internet;
- ter como objetivo o oferecimento de serviços que atendam uma variada faixa de modalidades de aplicações na Internet : negócios eletrônicos com consumidores no varejo (B2C), negócios-para-negócios (B2B), par-a-par (P2P) e comunicação com aplicações integradas de empresas (EAI);
- viabilizar a distribuição modular da aplicação de modo centralizado e descentralizado, em ambiente de aplicações que suportam fraco acoplamento na comunicação de aplicações dentro do ambiente de rede da empresa (Intranet) e fora do ambiente de rede da empresa (Extranet). Isto no que diz respeito a conectividade entre sistemas de aplicação;
- possibilitar a publicação de ofertas de serviços para um ou mais diretórios que sejam públicos ou privados, permitindo que potenciais usuários possam localizar os serviços publicados usando mecanismos padrões de busca, definidos pelas organizações de padronização;
- disponibilizar que os Web Services quando sejam requisitados possam estar sujeitos a autenticação, autorização e outras medidas de segurança[10].

Os requisitos anteriormente citados estão divididos, entre três componentes lógicos que são blocos básicos nos quais se mapeiam as operações necessárias e o entre os quais ocorrem os relacionamentos.

- Conteiner de serviço/ambiente de execução – o container de serviço comporta-se como o ambiente de execução do Web Service, hospeda e provê, disponibiliza para uso dos possíveis clientes de Web Services. Tipicamente em um ambiente de aplicação Web, é definido o container de Web Services como um ambiente de execução para os serviços de comunicação com os clientes no qual são expostas para acesso interfaces de componentes que fazem parte de alguma aplicação corporativa. A função do conteiner de serviço é facilitar a disponibilização e a administração dos Web Services. Adicionalmente, este ambiente também gerencia o serviço de fornecimento da descrição de Web Services e o registro destes nos diretórios de publicação. Existem, casos em que servidores Web de aplicações fornecem sistemas de serviço e APIs que podem ser usadas para interagir com container de Web Services;
- Serviços de registro – estes serviços hospedam os anúncios dos serviços publicados e comportam-se como corretores fornecendo facilidades para a publicação e o armazenamento da descrição de Web Services registrados pelos provedores, fornecedores de serviços. Adicionalmente definem o ponto comum e o mecanismo de acesso aos serviços publicados, os quais devem ser acessados pelo solicitantes para localizar os serviços requisitados;
- Serviço de entrega – comporta-se como o cliente do Web Service no ambiente de execução pois procura pelo Web Services no serviço de registro e em seguida invoca, utiliza e consome o Web Services desejado no serviço provedor. É representado por um módulo de apresentação para solicitação de serviços, expondo as interfaces apropriadas e os custos para a geração de conteúdo a ser entregue para uma variedade de possíveis clientes como aplicações, dispositivos de acesso, plataformas e outros[10].

Tendo como base a linguagem XML que constitui o padrão para Web Services e para as tecnologias dos blocos que fazem parte do núcleo básico da arquitetura de Web Services. Podemos afirmar que a arquitetura típica de um Web Services reside *de facto* na padronização das partes componentes, referenciadas canonicamente como WUST (WDSL, UDDI, SOAP Technologies), de modo que estas tecnologias estão coesamente relacionadas e comunicam-se entre si. Temos ainda a citar o padrão ebXML que colabora com especificações em partes do WUST provendo padrões para a condução e utilização de comércio eletrônico de modo global e integrado para que sistemas de negócios eletrônicos baseados em Web Services possam aderir a arquitetura proposta[10].

Entre os padrões e tecnologias baseadas em padrões bem definidos que compõem os blocos básicos do núcleo de arquitetura de Web Services, os principais e suas funções serão destacados a seguir :

- SOAP – é um modo padronizado de transmissão de dados e comporta-se como um serviço de mensagem que possibilita o serviço de invocação de mensagem e chamada entre o provedor do Web Services e o serviço solicitante, o consumidor. No padrão baseado na arquitetura ebXML está função é desempenhada pelo ebXML Messaging que provê maior confiabilidade na troca de mensagem entre serviço provedor e serviço solicitante;
- WSDL – reside no conteiner do Web Services e provê, fornece uma descrição padronizada do Web Services para que o solicitante possa consumi-lo. No padrão baseado na arquitetura ebXML esta tarefa é desempenhada pelo ebXML CPP/CPA que disponibiliza a descrição do serviço incluindo as especificações necessárias para o parceiro solicitante e os termos de acordo par uso do serviço;
- UDDI – é a estrutura que deve disponibilizar o mecanismo padrão para publicação e localização, descoberta de Web Services registrados e oferecidos, também pode comportar-se como dispositivo de armazenamento e repositório das descrições de Web Services em WSDL. No padrão baseado na arquitetura ebXML, o ebXML Registry & Repository (registro e repositório) disponibiliza facilidades para o armazenamento de descrições CPP/CPA para a colaboração entre aplicações de negócios.

Como podemos notar, Web Services são utilizados empregando-se padrões de protocolos da Internet e XML. Logo Web Services formam uma arquitetura construída sobre uma infra-estrutura padrão para a construção de aplicações distribuídas como serviços que podem ser publicados, localizados e acessados pela Internet por qualquer cliente que se adeque a estes padrões já estabelecidos[10]. Na Figura 2.5-12 a seguir, ilustramos a arquitetura de Web Services e os blocos do núcleo de construção, empregando conjuntamente os principais padrões.

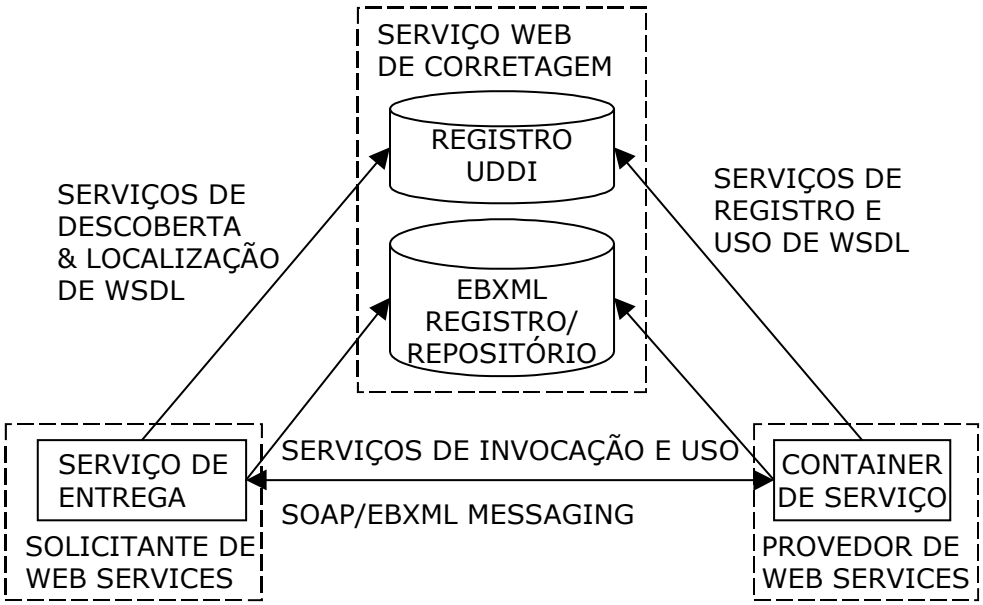


Fig. 2.5-12 arquitetura de Web Services e os blocos do núcleo de construção

2.6 - Modelo de comunicação de Web Services

De acordo com os requisitos funcionais e as especificações do problema a ser solucionado, a arquitetura de Web Services permite a implementação de dois modelos de comunicação. Os modelos de comunicação possíveis de implementação são o baseado em RPC síncrono ou o modelo de comunicação baseado em mensagem síncrono e assíncrono.

2.6.1 - Modelo de comunicação baseado em RPC

O modelo de comunicação RPC define a comunicação caracterizada pelo par solicitação/resposta que é o modo de troca de mensagens síncrono. O cliente envia a solicitação e aguarda até que a resposta seja enviada de volta pelo servidor, antes de continuar qualquer operação. Essa é uma implementação típica de outros modelos de computação distribuída como CORBA e RMI . O modelo de comunicação Web Services baseado em RPC apresenta alto grau de acoplamento, é implementado empregando-se objetos remotos que são utilizados pela aplicação cliente. Nesse modelo os cliente utilizam a capacidade de passar parâmetros em chamadas de métodos para os provedores de Web Service. Os clientes chamam os provedores de Web Services enviando parâmetros como valores, os provedores de Web Services executam os métodos solicitados e retornam o resultado para os clientes que podem ser desde valores atômicos até valores complexos, como objetos. Adicionalmente, para empregar este modelo de comunicação faz-se necessário que provedores de Web Services e

clientes solicitantes de serviços registrem-se e descubram-se, respectivamente, nos serviços de diretórios disponíveis[10]. A Figura 2.6.1-13 oferece uma representação gráfica, simplificada desse modelo de comunicação possível de ser implementado na tecnologia Web Services.

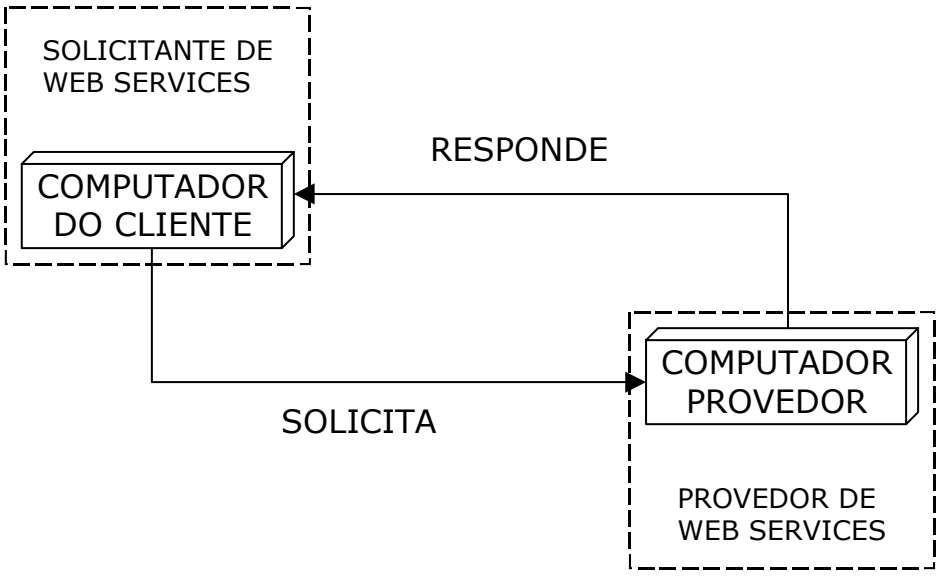


Fig. 2.6.1-13 modelo de comunicação baseado em RPC.

2.6.2 - Modelo de comunicação baseado em mensagem

Web Services podem ainda ser disponibilizados oferecendo modelo de comunicação baseado em mensagens. Por conceito esse modelo define baixo acoplamento e constitui-se em um modelo de comunicação dirigido a documento. O solicitante do serviço ao invocar o provedor do Web Service não tem a perspectiva de receber uma resposta[10]. Nesse modelo de comunicação, o cliente invoca o servidor do Web Service, envia diretamente ao mesmo um documento completo em vez de um conjunto de parâmetros relativo a chamada de algum método. O servidor ao receber o documento enviado pelo cliente, processa-o e poderá ou não enviar de volta uma resposta.[10]. Dependendo da implementação, o cliente poderá enviar ou receber documentos assincronamente de e para um Web Service baseado em mensagem, mas essas funcionalidades não podem ser executadas simultaneamente no cliente. Podemos ainda ter o modelo de comunicação baseado em mensagens síncrono, no qual o cliente envia solicitação para o servidor e aguarda até receber um documento como resposta[10]. A Figura 2.6.2-14 apresenta graficamente o modelo de comunicação baseado em mensagem comporta-se funcionalmente.

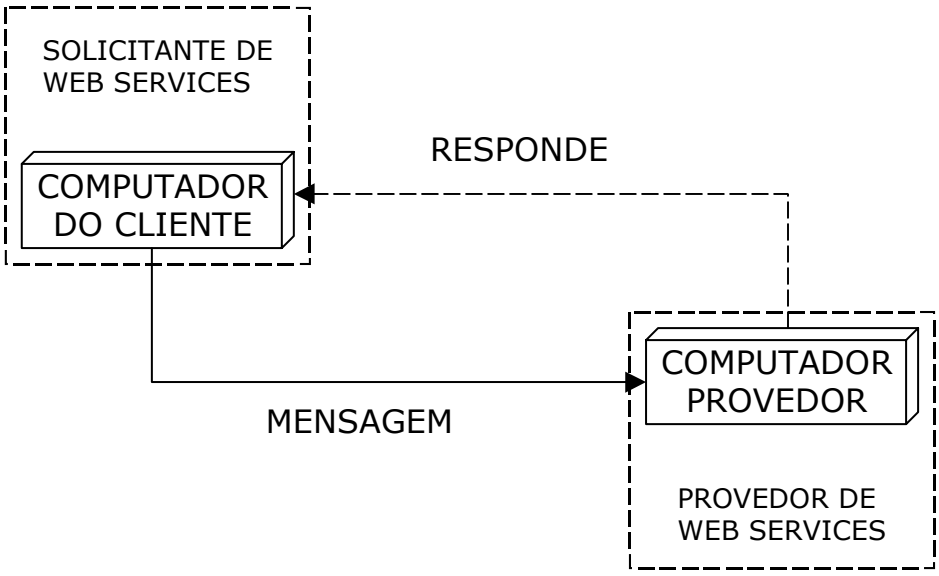


Fig. 2.6.2-14 modelo de comunicação baseado em mensagem.

2.7 – Modelo de implementação Web Services

O processo de implementação de Web Services é bastante similar aos processos de implementação de outros modelos de computação distribuída como CORBA e RMI. Entretanto, a diferença específica do modelo de implementação de Web Services está no fato de que os componentes do mesmo interagirão somente no tempo de execução, no instante em que os mesmos forem usados efetivamente, de modo dinâmico, através de protocolos padrão[10]. A seguir descreveremos os principais passos do modelo de implementação de Web Services, os números que seguem cada ponto da descrição, referem-se as etapas representadas na Figura 2.7-15 que será oferecida em seguida a descrição dos pontos :

- provedor de serviço cria os Web Services tipicamente como serviços baseados na interface SOAP para expor aplicações ou componentes de negócios. O provedor disponibiliza os Web Services em um container de serviço ou através de um ambiente de execução SOAP, e o torna acessível através da Internet. O provedor de serviço também disponibiliza a descrição dos Web Services no padrão WSDL para acesso ao serviço, a qual define como o cliente e o container devem interagir de modo consistente, expressando claramente a identificação e localização do serviço, operações disponíveis e o modelo de comunicação implementado (1);
- após a implementação dos Web Services, o provedor de serviços registra os mesmos, com base na descrição de serviço WSDL, no corretor de serviços tipicamente um registro UDDI (2);

- diretório de registro de Web Services, um registro UDDI armazena a descrição do serviço em formato padronizado, disponibilizando URL para a localização do WSDL no ambiente do provedor dos Web Services (3);
- solicitante do serviço pode então localizar os Web Services desejados a partir de pesquisas no registro UDDI. O solicitante dos Web Services obtém a informação necessária e o URL para identificar e acessar o provedor de Web Services (4);
- empregando as informações obtidas no registro UDDI, o solicitante do serviço, invoca o provedor dos Web Services e solicita a descrição dos serviços, no padrão, WSDL para os serviços registrados. Após isto o solicitante dos serviços cria um proxy cliente para que a aplicação tenha acesso aos Web Services estabelecendo comunicação com o provedor, através do SOAP (5);
- na seqüência, o solicitante dos Web Services comunica-se com o provedor e inicia a troca de dados ou mensagens, invocando os serviços disponíveis no container (6)[10].

Na Figura 2.7-15, a seguir, apresentamos de modo simplificado como e onde cada um dos pontos descritos anteriormente ocorrem. O número apresentado após a descrição dos pontos tem a função de facilitar a associação do texto com a imagem.

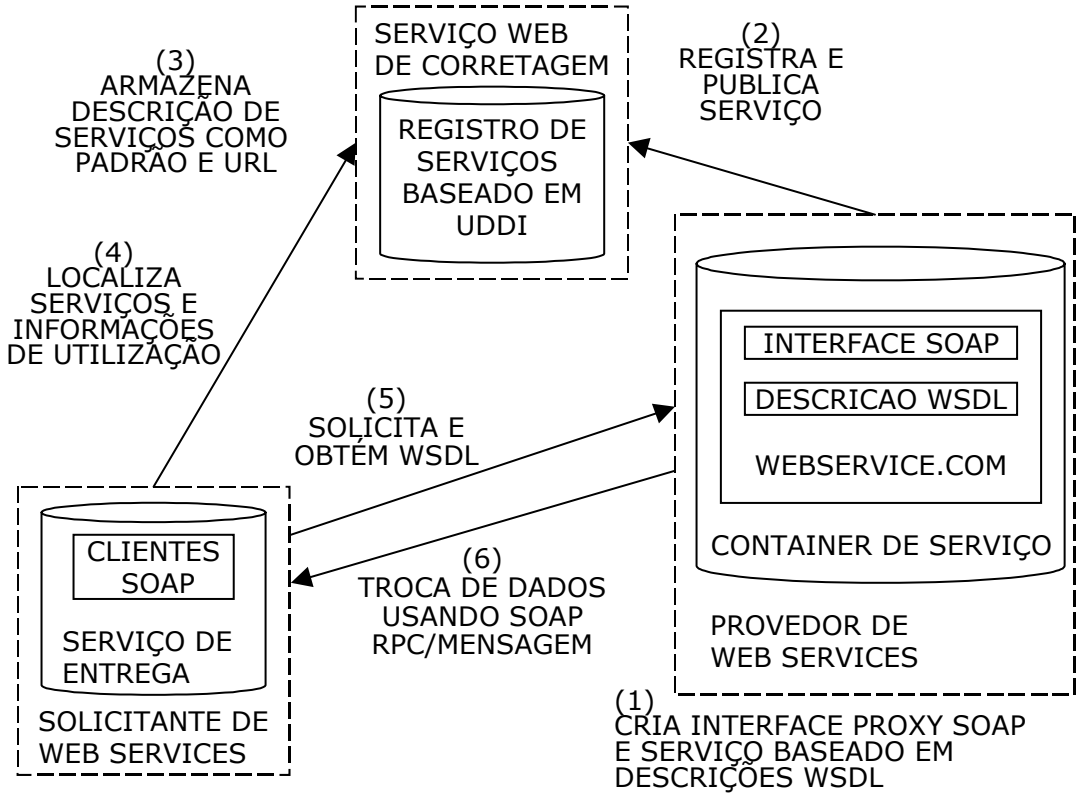


Fig. 2.7-15 modelo de implementação de Web Services.

Para os ambientes baseados em ebXML, os passos mostrados anteriormente guardam grande semelhança, porém temos a considerar algumas exceções como o ebXML registry and repository, ebXML Messaging e ebXML CPP/CPA que são usados no lugar dos padrões UDDI, SOAP e WSDL respectivamente[10]. A descrição e a figura aqui apresentadas focam enfaticamente na arquitetura de Web Services, tentando mostra de modo simplificado como esta funciona. Foram ignorados muitos aspectos críticos como implementação de segurança e qualidade de serviços.

2.8 – Modelo de emprego de Web Services na exposição de aplicações

Para o projeto e desenvolvimento de Web Services deve-se aplicar os mesmos princípios e técnicas usados para desenvolvimento de aplicações distribuídas. Um dos principais fatores diferenciais desta tecnologia é que um Web Service pode ser criado a partir de uma aplicação totalmente nova, criada para ser um Web Service ou pode ser implementado a partir de uma aplicação já existente, disponibilizando-se parte desta como um Web Service[10] em um processo de reaproveitamento de partes da aplicação como um serviço em ambiente de computação distribuída.

Com a tecnologia Web Services é possível expor-se aplicações existentes e legadas como serviços, aplicando-se a técnica do encapsulamento do núcleo das funções de negócio destas aplicações, cobrindo-as com a nova camada de nova tecnologia. As aplicações por baixo da camada de encapsulamento podem ser escritas e implementadas em qualquer linguagem de programação e estar funcionando sobre qualquer plataforma[10].

O modelo de implementação de Web Services constitui-se de modo geral em um modelo que provê a implementação de serviços orientados a interfaces com suporte a uma variedade de ambientes de aplicações de retaguarda que funcionam sobre os mais diferentes sistemas operacionais e plataformas[10]. Os passos genéricos para o desenvolvimento de Web Services, a partir de tais aplicações, são descritos a seguir.

- toma-se uma aplicação e analisa-se os seus potenciais componentes ou funções que podem ser encapsuladas em uma interface orientada a serviços usando SOAP e então expõe-se este componente ou função como Web Services, disponibilizando-o para acesso, em um container Web Service ou através de um ambiente de execução SOAP. Através da interface baseada em SOAP, o container de serviço manipula os acessos que chegam em SOAP no formato solicitação/resposta ou operações baseadas em mensagem e mapeia os métodos e argumentos para as aplicações de

negócio[10] que foram previamente encapsuladas e preparadas para responder, como sempre fizeram, a tais solicitações, só que neste contexto suas respostas serão envelopadas em SOAP;

- gera-se, automaticamente, a descrição, no padrão WSDL, dos Web Services disponibilizados e expostos, a qual será armazenada e disponibilizada no container de serviço. Esta descrição WSDL define o contrato de comunicação requerido para a invocação das interfaces baseadas em SOAP. Esta descrição WSDL também deverá ser publicada em um registro UDDI, como um serviço padrão que utiliza apenas algumas partes do WSDL e mais a URL para a localização do WSDL completo. As interfaces para a publicação no registro UDDI são geralmente disponibilizadas pelo container de serviço dos Web Services;
- localiza-se o serviço desejado empregando-se os mecanismos de busca (API para acesso aos registros UDDI) e obtém-se a descrição e a localização do provedor através da URL, isto é papel do solicitante. Em seguida conecta-se com o provedor do Web Service para obter o WSDL;
- utiliza-se o serviço exposto pelo provedor de Web Service, com o solicitante exercendo a capacidade de implementar interface cliente baseada em SOAP que atenda aos requisitos e parâmetros descritos no WSDL fornecido pelo provedor do Web Services que se deseja utilizar[10].

Os conteiners de Web Services, também chamados de container de serviços, e os ambientes de execução de Web Services geralmente disponibilizam ferramentas adequadas para a criação das interfaces baseadas em SOAP compatíveis com diversos tipos e ambientes de aplicações já existentes [10]. Isto é estas ferramentas tem a função construir a ponte para que uma aplicação seja exposta como um Web Services.

Além de produzir o código necessário para expor partes de aplicações existentes como Web Services as ferramentas também produzem automaticamente os descritores baseados em WSDL e esquemas automatizados para publicação do Web Services em UDDI [10]. Ilustramos este tópico com a Figura 2.8-16 que tem o objetivo de demonstra, de forma simplificada, como o modelo de Web Services pode expor aplicações diversas em um ambiente padrão de Internet com ênfase no ambiente do provedor de Web Services.

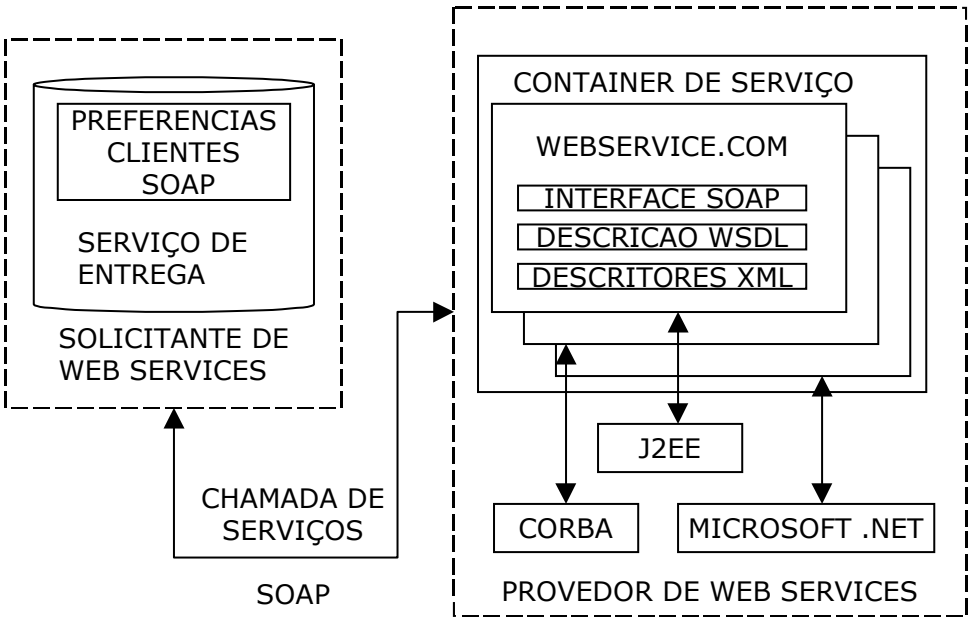


Fig. 2.8-16 modelo Web Services na exposição de aplicações.

2.9 - Limitações da tecnologia Web Services

Muitas das limitações da tecnologia de Web Services estão relacionadas com o ambiente no qual devem operar, a Internet. Este ambiente pode ser de certa forma considerado hostil, pois trata-se de um ambiente aberto[3]. Algumas das limitações estão relacionadas também com as tecnologias de Internet sobre as quais os Web Services estão fundamentados. Outras limitações são inerentes as próprias especificações elaboradas para os mesmos[16].

2.9.1 – Descoberta e localização

Para solucionar o problema de mudanças de local dos Web Services a indústria adotou o padrão de diretório denominado UDDI onde são publicados os endereços de acesso dos Web Services. Caso um Web Services tenha seu funcionamento alterado, com a adição de novas funcionalidades, as funcionalidades antigas devem ser mantidas para que aplicações dependentes dessas permaneçam em funcionamento e as novas devem ser adicionadas de modo a não interferir com as antigas[3]. Entretanto um novo WSDL deve ser gerado e disponibilizado para consumo, além da atualização necessária no UDDI.

2.9.2 – Confiabilidade no ambiente de Internet

O número de falhas e não respostas por mês ou por ano representam a medida de confiabilidade de um provedor de Web Services. Em outras palavras, confiabilidade refere-se a garantia e capacidade ordenada de entrega de mensagens por parte do serviço de entrega para os serviços solicitantes[3]. A confiabilidade de Web Services esta relacionada ou dependente do fato de os mesmos utilizarem HTTP. HTTP constitui-se um padrão de Internet, a melhor tentativa de operacionalidade, porém a melhor tentativa para entrega de resposta a solicitação de um serviço[3]. Com a adoção de HTTP não há garantia de que os pacotes serão entregues no destino e nem garantia de que estes pacotes serão entregues na ordem correta[3]. Uma tentativa de minimizar esse problema é a implementação de um novo protocolo derivado do HTTP, o HTTPR, onde R (reliable)[3]. Outras opções como os protocolos BEEP (Block Extensible Exchange Protocol), DIME (Direct Internet Message Encapsulation) podem ser adotados, porém não fazem parte dos padrões normalmente adotados na Internet e será necessário algum tempo para serem adotados[3]. Uma das soluções possíveis para o aumento da confiabilidade é o uso de filas de mensagens, mas deve-se mensurar o custo relativo ao aumento do tempo de resposta[3].

2.9.3 – Segurança

Alguns Web Services estarão disponíveis para uso público e não possuirão nenhum esquema de segurança, porém a grande aplicação de Web Services em B2B necessitará de criptografia e autenticação[3]. Como soluções para autenticação de usuários poderá ser realizada através de HTTPS utilizando-se SSL (Secure Socket Layer)[3]. O emprego de assinaturas digitais também poderá ser adotado para possibilitar autenticação[3]. O amadurecimento dessa tecnologia promove padrões direcionados para a aplicação de Web Services nas relações em que são exigidos níveis maiores de segurança, o padrão ebXML é um padrão criado para atender a esta limitação[10].

No aspecto da autenticação, considerando que esta deve identificar para o Web Services o cliente que o está acessando[16], não ha definição sobre qualquer padrão. A solução adotada é entregar esta tarefa para o container de Web Services. Esta opção propicia a criação de soluções proprietária e contraria a promessa de portabilidade entre fabricantes[16].

A garantia e o registro de que houve realmente comunicação entre cliente e provedor de Web Services é denominada de norepudiation[16]. Esta propriedade deve funcionar como um log de atividades entre os participantes da relação, entretanto, não existem padrões definidos sobre as responsabilidades das partes no que diz respeito a manutenção deste tipo de registro. Tratando-se de

característica indispensável par a adoção de Web Services, mais uma vez a industria toma a iniciativa e entrega esta tarefa para o container de Web Services e recai no lugar comum das soluções proprietárias de cada fabricante[16].

2.9.4 – Transações

Sistemas tradicionais que utilizam transações empregam a abordagem two-phase commit (confirmação em duas fases) [48] , na qual temos a primeira fase, onde todos os participantes são avisados para travar os recursos necessários, após o travamento confirmado por parte de todos, então, na segunda fase, efetiva-se a transação e destrava-se os recursos. Esta abordagem demostra-se adequada e funcional para ambientes controlados e fechados nos quais as transações tem curta duração[3]. Porém esta abordagem demonstra-se inadequada para ambientes abertos, como a Internet, e também quando tem que gerenciar transações de longa duração que podem consumir horas e até mesmo dias[3].

Faz-se necessários adequar-se as transações de longa e também de curta duração aos Web Services. Uma das soluções possíveis para esta limitação é a proposta do OASIS (Organization for Advanced Structured Information Standards) através do BTP (Business Transaction Protocol) que estende o método two-phase commit para as necessidades e características dos Web Services, garantindo as propriedades ACID (Atomic, Consistent, Isolated and Durable) [46] para as transações e transações non-ACID (que devem ter terminação coerente) utilizadas para transações de longo tempo que envolvem muitos participantes[3].

Outra alternativa para a abordagem two-phase commit é a "compensação" que baseia-se no pressuposto de que a transação iniciada esta sempre pronta para ser efetivada, mas seus efeitos e ações podem ser canceladas após sua efetivação[3]. Nesta abordagem, cada transação real possui uma transação de "compensação" associada que é composta por elementos que descrevem como a transação real deve ser desfeita fazendo com que seja possível o retorno ao estado anterior a sua efetivação[3]. Desta forma, caso algum dos participantes execute a sua transação de "compensação", todos os demais participantes são avisados para executarem as suas respectivas transações de "compensação", retornando ao estado anterior a efetivação da transação real. Os problemas com esta abordagem são : não satisfazem os requisitos ACID em todos os casos, sempre pode haver a possibilidade de falhas e transações projetadas para a abordagem two-phase commit devem ser redesenhadas para operar sob a abordagem da "compensação".

2.9.5 – Escalabilidade

Escalabilidade constitui-se na capacidade de aumentar-se os recursos e linearmente, idealmente falando, aumentar-se a capacidade de serviço, em termos de quantidade de atendimentos e velocidade de resposta[3].

A característica chave da escalabilidade de uma aplicação é a de que o aumento de carga sobre ela requer somente o aumento dos recursos sobre os quais ela funciona e nunca requer extensivas modificações na arquitetura da aplicação[3]. Em modelos de computação distribuída como CORBA, DCOM, RMI, aplicações legadas e EIS a escalabilidade está ligada intrinsecamente ao projeto e a construção da aplicação e dos componentes destas aplicações expostos como Web Services[10]. Esta característica pode causar limitações para este grupo de Web Services derivados destas tecnologias.

Considerando que é possível a exposição de EJB (Enterprise Java Beans) [8] como Web Services, pode-se emprega técnicas como balanceamento de carga e outros mecanismos que possibilitam aumentar a escalabilidade de Web Services oriundos de aplicações desta tecnologia[3].

2.9.6 – Desempenho

O desempenho de um Web Service é medido em termos de tempo de resposta e latência. O tempo de resposta representa o número de solicitações a Web Services respondidas em um determinado período. Latência é o tempo de ida e volta entre o envio de uma solicitação e o recebimento da resposta[3].

Temos a considerar na questão desempenho que o SOAP é de *facto* o protocolo padrão para Web Services e a desempenho do SOAP é degrado pelos seguintes fatores :

- SOAP emprega XML em vez do formato binário o que torna o tamanho dos dados cerca de 400% maior que os dados binários;

- a extração do envelope SOAP a partir dos pacotes SOAP requer esforço de processamento e torna-se caro em termos de consumo de tempo;

- parsing das informações XML contidas no envelope SOAP também requer esforço de processamento e consumo de tempo;

- a codificação da dados binários em um formato compatível com XML requer esforço de processamento e consumo de tempo;

- os parsers de XML são oferecidos com uma quantidade grande de facilidades, as quais não são necessárias ao SOAP, o que faz com que estes parsers necessitem de processamento intensivo e tornem-se lentos[3].

Existem algumas aplicações que empregam a estratégia de compressão do XML quando percebem que a utilização da CPU para a compressão é menor que a latência da rede. Esta aplicações utilizam versões reduzidas de parsers XML que possuem código de execução reduzido e ocupam pouca memória[3]. Entretanto a maioria dos parsers empregados para SOAP são baseados em DOM (Document Object Model) que exigem grande quantidade de memória[10] e demonstram-se lentos para a realização do parse de mensagens[3]. Entretanto o emprego de SAX (Simple API for XML) implementado para SOAP pode aumentar o tempo de resposta e reduzir a necessidade de memória[3].

Outro aspecto que influencia no desempenho dos Web Services é o fato de este, na grande maioria das implementações, estar disponibilizado sobre HTTP que foi originalmente projetado para que um servidor de páginas Web respondesse centenas de solicitações muito rapidamente, mas não mantivesse por muito tempo o estado da conexão entre o cliente e o servidor[16].

A cada solicitação, o HTTP inicia uma conexão nova entre servidor e cliente e a mantém somente enquanto está enviando os dados[16]. Tipicamente o servidor de HTTP, não mantém nenhum tipo de controle sobre o estado da conexão, de modo a possibilitar o rastreamento da navegação de um usuário. Porém, isto pode ser implementado através do controle de sessão ou de cookies. Esse é um comportamento que faz com que HTTP tenda a ser muito transacional[16]. Esta característica obriga ao servidor perder tempo, iniciando e terminando uma conexão para cada solicitação do cliente[16] que no caso de Web Services, tenta manter um "diálogo" contínuo durante a utilização de métodos.

2.9.7 – Modelo de cobrança e precificação

Web Services comerciais irão necessitar de um modelo segundo o qual devam ser remunerados pelos serviços prestados[3]. Uma das propostas é a de que se crie serviço intermediário que possa medir o quanto um determinado cliente utilizou de um Web Services e apresente a este um montante relativo a cobrança por esta utilização[3]. Essa constitui-se apenas em uma das propostas até agora apresentada. Na tentativa de solucionar o problema, alguns provedores de Web Services implementam e disponibilizam soluções de cobrança proprietárias, quebrando o princípio da portabilidade[16].

2.9.8 – Disponibilidade

Usuários de Internet estão cientes de que um site não está 100% do tempo disponível[16]. Esta é uma característica da Internet que afetará os sites de Web Services. Pode ocorrer de um servidor está rodando uma transação e o servidor de Web Services participante não estar disponível. Para situações assim são necessários mecanismos que tentem novamente concluir a transação, quando o servidor estiver disponível, ou a desfaçam sem grandes transtornos[16]. Alguns dos novos protocolos, como JMS (Java Messaging System), suportados pelos Web Services podem lidar com este tipo de situação de forma automática, porém protocolos como HTTP e outros, não dispõe de tal capacidade[16].

2.9.9 – Adequação de requisitos

Quando cria-se um Web Services genérico para atender a uma variedade de clientes, necessita-se considerar um conjunto especializado e limitado de especificações e requisitos[16]. Alguns clientes poderão necessitar de algumas pequenas facilidades as quais não são necessárias a mais nenhum outro cliente. Web Services podem ser vistos como uma tecnologia que produz "um único tamanho adequado a muitos clientes"[16]. Para negócios que não se adaptam a este tipo de abordagem outras soluções devem ser consideradas.

2.9.10 – Interfaces imutáveis

No segmento de Web Services que objetivam atender a um conjunto diversificado de clientes uma das principais atitudes a ser evitada é a mudança em qualquer dos métodos e nos seus

parâmetros esperados pelas aplicações dos clientes[16]. Pode-se criar novos métodos, adicionar facilidades a métodos antigos, porém as versões anteriores devem ser mantidas sob pena de que as mudanças nestas versões possam vir a provocar falhas nas aplicações dos clientes[16].

É possível descobrir-se que um dos métodos disponibilizados nos Web Services contém erros e está funcionando incorretamente. Nessa situação, temos duas possibilidades : se o erro for interno e não exigir mudanças da interface, dos parâmetros e dos métodos externamente, temos de descobrir quem utiliza o Web Service e comunicar sobre o mal funcionamento e da correção. Mas se por outro lado, a incorreção necessitar de mudanças na interface, então os programas dos clientes deverão ser modificados mantendo-se ainda a necessidade de avisá-los sobre a incorreção do Web Service[16]. Isto ocorre devido ao alto acoplamento entre provedores e usuários de Web Services[16].

2.9.11 – Garantia de execução

O protocolo HTTP não é um confiável, não garante entrega e nem garante reposta. Se sua aplicação de Web Services possui este requisito existem três possibilidades a considerar : escreva código que garanta este comportamento, escolha um intermediário que possa oferecer esta garantia ou adote uma das novas tecnologias disponíveis para Web Services como JMS[16].

2.10 – Conclusão

A Internet representa um meio de comunicação poderoso que deve ser utilizado para a interação entre programas. Discutimos as muitas tecnologias disponíveis para a computação distribuída e sua possível aplicação na Internet como meio de suporte para a interação entre sistemas corporativos.

Diante das duas principais fraquezas das tecnologias apresentadas que produzem alto acoplamento entre fornecedor e consumidor de recursos e adicionam "camadas" de artefatos estranhos a Internet e necessários para a adoção de qualquer uma delas. Surge como alternativa os Web Services.

Baseados em XML e em tecnologias abertas como WSDL, UDDI e SOAP, Web Services apresentam-se como uma das possibilidades para a implementação do compartilhamento de recursos e a integração entre sistemas corporativos empregando a Internet como meio de comunicação. Suas três principais vantagens estão embasadas sobre o fato de funcionarem sobre padrões estabelecidos na Internet (XML, HTTP, TCP/IP), baixo acoplamento e heterogeneidade entre as partes

que pretendem interagir. Porém, Web Services possuem limitações e comportamentos que devem ser considerados quando da sua utilização, pois trata-se de uma alternativa não recomendada para alguns casos com características bem definidas.

No próximo capítulo, discorreremos em detalhes sobre as tecnologias principais que compõem o núcleo de Web Services, demonstrando conceitos de constituição do protocolo de transporte SOAP, da linguagem de descrição de Web Services WSDL e do diretório para a divulgação de serviços disponibilizados UDDI.

Capítulo 3 – Tecnologias Envolvidas em Web Services

3.1 - Introdução

Muitas definições podem ser dadas para Web Services, entre elas podemos defini-los como arquitetura para sistemas distribuídos que possibilitam aplicações orientadas a objetos a disponibilizarem serviços entre si [11]. Outra forma de definir Web Services são qualquer arquitetura que envolva o transporte de documentos XML entre sistemas de plataformas diferentes [11]. Não importa qual definição de Web Services seja adotada, não poderá haver dúvidas de que a definição adotada emprega SOAP, WSDL e UDDI [11] e todos estes estão baseados em XML.

Partindo de baixo para cima, apresentamos primeiramente XML como base sobre a qual se construiu a tecnologia Werb Services. A seguir descrevemos o conjunto de tecnologias empregadas para formar a tecnologia Web Services com ênfase nas três principais partes : SOAP, WSDL e UDDI que juntas denominam-se WUST (WSDL-UDDI-SOAP Tecnologies).

Evoluindo a partir desse conjunto, outras tecnologias acrescentaram sofisticação e ampliaram o escopo de aplicação dos Web Services. Nesse contexto apresentamos as principais entre estas.

3.2 - XML

A inspiração para o projeto de XML veio de duas fontes distintas : SGML (Standard Generalized Markup Language) e HTML (Hypertext Markup Language)[12]. A versão 1.0 de XML foi disponibilizada pelo W3C (World Wide Web Consortium) em 10 de fevereiro de 1998, porém o projeto

da linguagem foi iniciado em meados de 1996, com objetivo de produzir mecanismo simples e extensível para representação textual de informação estruturada e semi-estruturada.

O conceito de marcadores generalizados (GM), utilizado em outras áreas por décadas, emprega tags (marcadores) para identificar pedaços de informações. Marcadores são nomes colocados entre os sinais de menor (<) e maior (>) respectivamente. Existem dois tipos de marcadores, os iniciais, que são colocados entre os sinais de maior e menor, como em <inicio> e os marcadores finais, que devem ser iguais aos iniciais, incluindo a barra (/), depois do sinal de menor, como em </inicio>. A inovação do conceito de marcadores generalizados está na necessidade de que a informação a ser sinalizada (marcada) tem que estar entre o marcador de inicio e o marcador de fim. A noção de marcador inicial e final possibilita o aninhamento, o que permite que a informação seja estruturada em uma ordem hieráquica[12].

A linguagem SGML, complexa e de implementação cara, foi ratificada pela ISO em 1996, sendo adotada somente por grandes usuários[12]. XML é similar a SGML e preserva o conceito de marcadores generalizados, partindo de uma especificação que pode ser considerada simples, mantém a possibilidade de extensão[12]. Assim muitas especificações são construídas sobre XML para estender suas capacidades e possibilitar o seu emprego em uma grande quantidade de cenários, inclusive em Web Services[12].

3.2.1 – Aplicações de XML

De modo genérico, existem duas grandes categorias de aplicação do XML : a primeira relacionada a aplicações centradas em documentos e a segunda em aplicações centradas em dados[12]. XML foi rapidamente adotado em sistemas de publicação como mecanismo para representação de documentos semi-estruturados entre os quais podemos citar manuais técnicos, documentos legais e catálogos de produtos[12]. O conteúdo desses documentos é tipicamente montado para o consumo humano, porém pode ser processado por muitas aplicações antes de ser apresentado em sua forma final[12]. O elemento principal dessa categoria de documento são textos semi-estruturados marcados em seções, capítulos, volumes e demais divisões que compõem uma publicação.

As aplicações de XML centradas em dados produzem informações textuais altamente estruturadas (hierarquizadas) representando bancos de dados relacionais, informações sobre transações financeiras, estruturas de dados de linguagens[12] e outras informações provenientes de fontes de dados estruturadas e organizadas com base em hierarquia. Estes dados são tipicamente produzidos por

aplicações para consumo de aplicações. A natural habilidade de XML para aninhar e repetir marcadores o torna a escolha adequada para este tipo de dado[12]. Em aplicações centradas em dados o emprego de XML possui características bem particulares em relação as aplicações centradas em documentos, entre estas podemos citar :

- quantidade de marcadores é elevada;
- não há textos longos nos marcadores;
- são incluídas informações produzidas pelo sistema de forma automática;
- marcadores são organizados de modo altamente estruturado;
- marcadores são empregados para descrever o significado da informação e não como ela deve ser apresentada[12].

Neste contexto de categorias genéricas de aplicações XML, temos a considerar o aspecto do ciclo de vida do documento ou seja temos que considerar a volatilidade da informação.

Tipicamente, documentos XML para consumo humano como manuais técnicos, papers de pesquisa, documentos legais e outros possuem longo tempo de vida útil porque a informação que guardam pode ser usada por um grande período de tempo[12].

Por outro lado, a grande maioria dos documentos XML centrados em dados possuem tempo de vida muito curto[12] pois são produzidos a partir de dados estruturados que podem ser modificados a cada milisegundo, de modo que a aplicação pode produzir o resultado de uma consulta sobre uma tabela em um banco de dados e entrega-lo como um documento XML, no momento seguinte, a consulta pode produzir um resultado diferente, pois alguma das tuplas [47] foi excluída. Web Services empregam documentos XML centrados em dados[12].

3.2.2 – Instâncias XML

A estrutura e a formatação de XML em um documento XML, deve seguir as regras de sintaxe para instâncias XML. O termo instância é explicitamente usado aqui para distinguir a diferença entre o uso particular de XML da sua especificação[12]. Neste caso estaremos descrevendo as partes possíveis e mais gerais de um documento XML. Entre estas partes temos : prólogo do documento, elementos e atributos.

O prólogo do documento XML constitui-se parte opcional que deve anteceder o elemento raiz o qual contém o documento em si. Pode desempenhar três funções : identificar o documento como um documento XML, incluir algum comentário sobre o documento e incluir meta-informação sobre o conteúdo do documento.

Um documento XML pode ser identificado como documento através de uma instrução de processamento PI (processor instruction) que constitui-se em diretiva especial com instruções para a aplicação que vai processar o documento. As instruções de processamento devem ser colocadas entre menor e interrogação (<?) e maior e interrogação (?>). Para identificar o documento emprega-se a PI padrão xml, que indica a versão do XML usado e, opcionalmente, o conjunto de codificação de caracteres empregados, assim teremos o identificador do documento :<?xml version=”1.0” encoding=”UTF-8”?>[12].

O prólogo pode ainda conter comentários referentes ao documento, estes devem ser inseridos sempre entre menor, exclamação e dois sinais de menos (<! - -) e maior e dois sinais de menos (- - >), comentários podem estender-se por várias linhas, porém não podem ser aninhados[12]. As meta-informações que podem ser inseridas no prólogo referem-se a possíveis regras de formação do documento segunda as quais o mesmo deve ser construído.

Um par de marcadores (inicial e final) é denominado tecnicamente, em XML, de elemento[12]. Todo marcador inicial deve conter um marcador final e vice versa. Tudo que for encontrado entre dois marcadores (inicial e final) é considerado conteúdo, incluindo marcadores aninhados, texto, comentários e outros componentes do XML. Os nomes de elementos seguem padrão similar ao empregado para linguagens de programação no que diz respeito aos nomes de variáveis : podem conter caracteres (0-9 A-Z a-z), sublinhado (_), hífen (-), dois pontos (:) mas devem necessariamente iniciar com uma letra[12]. A especificação de XML indica que o mesmo é sensível a letras maiúsculas e minúsculas. Esta mesma especificação classifica o conteúdo de um elemento XML em três tipos : somente elementos, misturado e vazio. O conteúdo somente elementos é constituído somente de elementos aninhados, nenhum espaço em branco separando elementos é considerado. Como conteúdo misturado temos qualquer combinação entre texto e elementos dentro de um mesmo elemento[12]. Finalmente o conteúdo vazio ocorre quando um elemento está presente e não há conteúdo para o mesmo, quando isto ocorre podemos representar o elemento com conteúdo vazio de duas formas : <vazio></vazio> ou <vazio/>.

Não encontramos na especificação XML definição formal para conteúdo somente composto por texto, porém a utilização continuada da linguagem, fez com que este tipo de conteúdo

fosse denominado de conteúdo de dados, mas tecnicamente, este tipo de conteúdo classifica-se como misturado[12]. Em aplicações de XML orientadas a dados o conteúdo misturado, que é composto de texto e elementos, nunca será visto[12], pois esta espécie de aplicação produz informação altamente estruturada. Em um documento XML temos o conceito importante de elemento raiz, todo documento XML possuí o elemento raiz que está no topo da hierarquia[12] e dentro do qual podem haver outros elementos, texto ou o mesmo pode estar vazio. O elemento raiz implica que deve haver apenas um elemento no nível maior, mais alto e mais externo da hierarquia. A presença de mais de um documento raiz, invalida o documento XML.

Dentro das especificações de instância do XML, temos a considerar o recurso de utilização dos atributos que são opcionais para os marcadores iniciais. Um atributo constitui-se de par ordenado, nome=valor que pode estar presente no marcador inicial contendo alguma informação[12] útil para a aplicação. As regras para a formação de nomes dos atributos são as mesmas empregadas na formação de nomes dos elementos. O emprego de atributos requer que os valores destes estejam contidos entre aspas, simples ou duplas[12], assim o par ordenado nome=”valor” ou nome=’valor’ pode ser empregado em qualquer elemento, sempre no marcador inicial. A especificação XML reserva para si os atributos iniciados em xml: [12], como exemplo desta família de atributos podemos citar o atributo xml:lang=”en” que neste exemplo indica o idioma inglês para o elemento do qual faz parte.

O emprego de atributos não obriga as aplicações de XML a reconhecer, processar e reagir com base nos atributos fornecidos junto com os elementos. A principal razão da identificação dos atributos em XML é que os mesmos são endereçados para casos de usos particulares, nos quais as aplicações estão preparadas para lidar com estes atributos e reagir de modo programado aos mesmos. Uma padronização ou formalização generalizada dos mesmos poderia adicionar incompatibilidade entre aplicações[12]. Na ausência de meta-informação sobre o documento XML, os atributos são considerados texto sem significado que acompanham os elementos[12] isto os torna perfeitamente compatíveis com aplicações de XML normais. Uma utilização dos atributos é a capacidade que estes tem de criar ligações entre informações (elementos) do mesmo documento, evitando a redundância de dados.

Completando a instanciação de XML temos a considerar o mapeamento de caracteres. Valores de atributos, texto e espaços em branco entre os marcadores devem seguir um pequeno, porém rigoroso, conjunto de regras[12] que podem ser discorridas em três grupos : a codificação, espaços em branco e entidades.

A codificação trata do mapeamento, da correspondência entre os caracteres que compõe o documento XML em questão para um conjunto de caracteres definidos. Por exemplo, pode-se mapear os caracteres de um documento XML para o conjunto de dados UTF-8, o importante é que o documento XML seja compatível com o conjunto de caracteres indicado para a codificação[12]. Qualquer caracter que não possa ser mapeado deve ser identificado como caracter de referência. A seqüência de escape (ESC) empregada por XML utiliza "é comercial" (&) como seu início e ponto e vírgula como final (;). A sintaxe para este tipo de caracter é composta de "é comercial", seguido por sustenido ( ), seguido pelo código decimal do caracter a ser mapeado ou o X minúsculo seguido pelo código hexadecimal, no final deve vir o ponto e vírgula (;), assim para mapear o caractere de 8 bits, 128 para a codificação UTF-8, podemos usar o formato XML &x80;[12].

Por razões relacionadas com a orientação a documentos, que XML herda de SGML, não existem meios de se incluir os caracteres de controle de espaço de 0 até 7, 9, 11, 12 e do 14 ao 31 (tipicamente conhecidos como controle de espaços não brancos em ASCII) em documentos XML. Não há como mapear estes caracteres no modo convencional e a presença dos mesmos causa problemas na manipulação de strings que os contenham[12].

Espaços em branco devem ser tratados de maneira particular em XML. Isto ocorre como legado de aplicações centradas em documentos[12]. Duas regras que devem ser definidas de maneira clara quanto a presença de espaços em branco em documentos XML. A primeira regra impõe que não existe quebra de linha em documentos XML, todo os CR (carriage return) e LF (line feed) devem ser retirados, o processador XML enxerga o documento como uma grande linha. A segunda regra diz respeito ao tratamento de espaços em brancos que pode ser significativo ou insignificante[12]. Aplicações orientadas a dados, como no caso de Web Services, são muito pouco afetadas pelo tratamento de brancos[12].

Todos os processadores XML devem reconhecer um conjunto padronizado de caracteres que definem entidades (componentes de documentos XML) e mapear estes caracteres de modo a não permitir que ocorra confusão com os caracteres que definem marcadores[12]. Estes caracteres são : menor (<), maior (>), "é comercial" (&), apóstrofo e aspa simples ('), aspa dupla (") que devem ser substituídos, respectivamente, pelas seqüências de caracteres equivalentes : menor (&lt), maior (&gt), "é comercial" (&amp), apóstrofo e aspa simples (&apos), aspa dupla (&quot)[12].

3.2.3 - XML Namespaces

Uma das mais importantes propriedades dos documentos XML é que os mesmos podem ser compostos, unidos para criar um novo documento. Este é um dos mais básicos mecanismos para o emprego da reusabilidade em XML[12]. A simples composição de documentos a partir de outros, produz dois problemas : reconhecimento e colisão[12]. O problema de reconhecimento diz respeito a possibilidade do processador XML reconhecer e separar que elementos do novo documento pertencem aos documentos que o originaram. O problema da colisão ocorre quando os documentos que foram empregados para dar origem a um terceiro, possuem os mesmos elementos ou seja elementos com nomes repetidos e recorrentes em mais de um documento usado para compor o documento final. Esses problemas podem não existir em pequenas aplicações de XML, pois a semântica de cada elemento reside na aplicação que o processa[12]. Entretanto, em grandes aplicações e a medida que muitas sistemas precisam manipular documentos compostos por outros os problemas de colisão e reconhecimento crescem e faz-se necessário a distinção clara entre os elementos XML[12]. A especificação XML Namespace tem a finalidade de colocar ordem no caos[12].

O mecanismo de XML Namespace endereça os dois problemas básicos da composição de documentos XML. O problema da colisão na composição cresce por causa da probabilidade de elementos com nomes comuns serem usados em documentos de diferentes tipos[12] o que ocorre com maior freqüência em cadeias de valor verticais. Este problema pode ser solucionado através da qualificação de elementos XML que resume-se em adicionar aos elementos XML um identificador que deve ser único entre os documentos que estão sendo usados na composição do novo documento[12]. Em outras palavras temos : nome qualificado (abreviado em inglês para Qname) = identificador de Namespace + nome local do elemento[12].

No caso do problema de reconhecimento, na composição de documentos XML, o mesmo ocorre por não haver bons mecanismos para identificar todos elementos que pertencem a um documento[12]. Com a introdução dos qualificadores o problema é resolvido de modo simplificado pois todos os elementos que possuem o mesmo qualificador fazem parte do mesmo documento e são considerados como estando juntos[12]. Como identificadores (qualificadores) a especificação XML Namespace emprega URIs (Uniform Resource Identifiers) que são descritas na RFC 2396[12]. Estes identificadores não são práticos de serem manipulados por humanos, porém são muito úteis[12]. Pode-se ainda empregar localizadores URI, também conhecidos como URL (Uniform Resource Locators) ou ambos como identificadores de elementos em documentos XML.

Como URIs são longos e quase sempre contem caracteres considerados inválidos pelas regras do nomes dos elementos XML, a sintaxe de inclusão para XML Namespace em documentos deve seguir duas normas básicas[12] : a primeira, um namespace deve estar associado a um prefixo que constitui-se em um nome contendo somente caracteres considerados válidos pela regra de formação de nomes de elementos XML, com exceção dos dois pontos (:)[12]. A segunda, nomes qualificados são obtidos a partir da combinação do prefixo, mais dois pontos (:) mais o nome do elemento[12], como em prefixo:elemento_local. Adicionando prefixo para cada elemento do documento, a legibilidade do mesmo diminui sensivelmente e o tamanho do documento aumenta[12], o que envolve mais espaço de armazenamento e esforço de processamento para manipular o documento. Entretanto o mecanismo XML Namespace permite que seja utilizado o namespace default (padrão) do documento. Com este dispositivo, os elementos que não forem informados com prefixo, recebem, automaticamente, o prefixo definido como padrão[12] e o processador XML pode identificar o namespace de todos os elementos presentes no documento.

Os atributos também podem possuir namespace associados a eles[12]. Esta facilidade foi implementada para permitir estende-se a capacidade de informação dos elementos, sem que fosse necessário realizar mudanças diretamente na estrutura do documento[12]. Namespaces aplicados a atributos possuem casos específicos de uso nos quais são adequados e devem ser empregados, porém o seu uso como recurso corriqueiro é desaconselhável. O exemplo de uso é o emprego de namespace em um atributo que deva ser processado por uma aplicação e ignorado por outra, este atributo pode ocorrer ou não em um elemento e quando ocorrer, deve ser ignorado por uma aplicação e processado por outra. Nesse cenário recorre-se ao namespace para este atributo.

3.2.4 – Data Type Definitions (DTD)

Herdados da SGML os DTDs da XML são extremamente simplificados em relação aos originais[12]. Constituem-se em uma facilidade opcional do XML e podem ser definidos como mecanismo que associa a um documento XML, conjunto de regras sobre a formação e a estrutura do documento, bem como regras sobre elementos e atributos que podem fazer parte do documento indicando a ordem e o local em que devem aparecer[12].

Os DTDs introduzem os conceitos distintos de documentos XML bem formados e válidos. Por documento bem formado entende-se todo documento XML que obedece as regras de sintaxe XML, que pode ser lido, processado sem erros básicos associados ao parsing (leitura de documento XML percorrendo marcadores iniciais e finais) tais como caracteres inválidos, ausência de

marcadores iniciais e finais, múltiplos marcadores com mesmo nome entre outros que podem ocorrer. A especificação XML indica de modo mandatário que caso algum erro de má formação no documento XML seja detectado o parse deve provocar erro grave, não recuperável e a leitura desse documento deve ser encerrada imediatamente[12]. Essa característica rigorosa proporciona a visão geral de dois focos possíveis para softwares que processam documentos XML : o foco na estrutura lógica direcionado para verificar o que cada marcador significa e o foco na estrutura física, direcionado para a verificação da sintaxe do documento[12].

Na maioria das aplicações que utilizam documentos XML o fato de ser bem formado não é o suficiente[12] como garantia de que os mesmos podem ser processados e cumprir a função para a qual foram criados. Um documento XML bem formado produzido por um programa de contabilidade não poderá ser processado por outro programa que espera um documento XML de nota fiscal. Nesse contexto temos a noção de validade. DTD constitui-se em mecanismo declarativo que pode automatizar a validação de conteúdos XML, quando estes são processados pelo parser[12], com a flexibilidade de que as aplicações XML podem limitar, direcionar o nível, a quantidade, a precisão de validação que desejam[12], sem que haja a necessidade de esforço de programação ou alteração de software.

Para manipular a checagem de validação, o mecanismo DTD pode implementar regras nos seguintes aspectos de um documento XML : identificação de elementos que podem estar presentes em um documento; identificação da ordem e da relação entre elementos (estrutura do documento) e identificação de atributos de todos os elementos bem como a determinação de onde estes atributos são obrigatórios ou opcionais[12].

Um dos modos para visualizar um documento XML definido por um DTD é através da construção de uma árvore hierárquica resultante da combinação de elementos e atributos. Os DTDs podem definir a relação entre elementos aninhados e suas ocorrência no documento. Temos os seguintes mecanismos de indicação de elementos dentro de um documento XML : interrogação (?) para elementos que podem ocorrer zero ou uma vez; asterisco (*) para elementos que podem ocorrer zero ou mais vezes e o sinal de mais (+) para elementos que devem ocorrer, obrigatoriamente, uma ou mais vezes[12].

Todos os elementos presentes em uma definição DTD, pertencentes a estrutura do documento e presentes na árvore que representa este documento estão associados a um grupo modelo que identifica a seqüência e a multiplicidade do conteúdo do elemento[12]. Existem dois tipos de seqüências : seqüência e escolha. No tipo seqüência podemos definir a ordem exata na qual os filhos de

um elemento devem aparecer. Emprega-se a vírgula para definir a seqüência, no grupo modelo (A, B, C) a seqüência indica que o primeiro filho é o A, o segundo é o B e assim por diante.

No caso da escolha, defini-se que elemento deve aparecer de modo exclusivo em detrimento dos demais. Utiliza-se a barra vertical (|) para separar os elementos que podem ocupar o lugar, mas somente um deles ocupará o lugar, no grupo modelo (A | B | C) a escolha deve ser A ou B ou C, um dos três[12]. Podemos ainda aninhar seqüências e escolhas para obter os resultados desejados em uma validação de documentos XML.

DTDs representam grande avanço na validação de documentos XML, evitando esforço de programação. Porém existem deficiências em DTDs que limitam sua aplicação em um grande números de aplicações principalmente as orientadas a dados. Entre estas limitações podemos citar : os arquivos de validação DTDs não são escritos em XML, tornando sua manipulação e processamento complexo; DTDs foram projetados antes dos XML Namespaces e não há muita aderência entre DTDs e namespaces que são fundamentais para aplicações orientadas a dados; os grupos modelos demonstram grande restrições, em particular no que se refere ao ordenamento de filhos; DTDs não comportam noções sobre tipos de dados, o que impossibilita a representação de estruturas de dados oriundas de linguagens de programação[12].

Devido as limitações anteriormente citadas e outras, protocolos de Web, como o protocolo SOAP, fundamental para Web Services, proíbem o emprego de DTDs para a definição de estruturas de documentos[12]. Os DTDs foram empregados para a realização de contratos entre partes que necessitavam trocar documentos XML com certo grau de formalização e confiabilidade[12]. Colaborando de modo importante para a adoção da XML e seu amadurecimento.

3.2.5 – XML Schemas

Para resolver as limitações impostas pelos DTDs em questões como integração com namespaces, projeto modular de vocabulário, modelo flexível de conteúdo e forte integração com aplicações orientadas a dados, o W3C apresentou em março de 2001 a versão final da especificação XML Schema[12] que trata-se de mecanismo, uma meta-linguagem, em XML, para descrever a estrutura de documentos XML e o mapeamento da sintaxe XML para tipos de dados[12]. Em uma frase podemos descrever XML Schena como poderosa, porém complexa[12]. Torna-se poderosa por permitir muito mais precisão e especificidade na definição do conteúdo de documentos XML e em virtude disso também torna-se mais complexa[12].

A especificação do XML Schema apresenta-se em três documentos : Parte 0 – Primer que trata-se de documento para consumo geral, pois não é normativo e composto por um grande número de exemplo, Parte 1 – Structures focado de modo técnico e rigoroso em documentos orientados a aplicações e Parte 2 – Datatypes procurar responder, de modo normativo e preciso, as necessidades de mapeamento das aplicações orientadas a dados, associando a sintaxe XML com esquemas de tipos de dados[12].

A maior diferença entre XML Schemas e DTDs está no fato de que o primeiro é expresso em XML[12]. Este fato elimina a necessidade dos programas de parser manipularem outra sintaxe (como no caso dos DTDs) e produz o ganho do poder de XML. Como não poderia deixar de ser o vocabulário do XML Schema também é autodefinido empregando XML [12].

Outra evolução importante do XML Schema, em relação aos DTDs, é que o mesmo foi projetado considerando a existência e a necessidade de aderir fortemente aos namespaces[12]. Neste aspecto, todos os elementos pertencentes a especificação do XML Schema são prefixados com : xsd: . O nome dos prefixos não é importante, mas no caso do xsd: (que vem das iniciais de XML Schema Definition) constitui-se convenção rígida. Esse prefixo é associado com a URL http://www.w3c.org/2001/XMLSchema namespace que identifica o documento de especificação do W3C Recomendation of XML Schema[12].

Na montagem de um XML Schema, temos a necessidade de indicar um namespace para os elementos do schema (XML Schema definido como um esquema particular de validação) que esta sendo montado a fim de que se possa distinguir os elementos do schema que esta sendo montado dos elementos que são prefixados com xsd:. Além destes dois namespaces, faz-se necessário indicar o namespace dos documentos que deverão ser validados no atributo targetNamespace[12].

Todos os esquemas definidos devem iniciar sempre com <xsd:schema> e finalizar com </xsd:schema>. Pode-se incluir elementos de anotação (xsd:annotation) e documentação (xsd:documentation) para se adicionar informações auxiliares ao schema[12] de modo a torna-lo mais claro e documentado.

O mecanismo de associação, ligação de XML Schemas a documentos conta com grande flexibilidade. Documentos não precisam estar associados a XML Schemas, uma aplicação pode ser configurada com antecedência para empregar determinado XML Schema quando processar um certo tipo de documento XML[12]. Alternativamente, existe um poderoso mecanismo para associação de

documentos XML com XML Schemas. O prefixo xsi: por convenção vem das iniciais de XML Schema Instance que possui significado especial quando indica como namespace http://www.w3c.org/2001/XMLSchema-instance definindo o numero de atributos que fazem parte do XML Schema especificado[12]. Estes atributos podem ser aplicados aos elementos na instância do documento produzindo mais informações para o processador do XML Schema[12].

A conexão entre o documento XML e o XML Schema que o valida é propiciada pelo atributo xsi:schemaLocation que deve conter dois valores : o primeiro é o namespace do XML Schema a ser usado e o segundo é a localização do XML Schema para que o mesmo possa ser acessado[12].

O segundo valor é uma URL porém aplicações especificas podem empregar outros tipos de valor como por exemplo um identificador para um repositório de XML Schemas ou um nome de XML Schema conhecido da aplicação[12]. Caso o documento a ser validado pelo XML Schema empregue mais de um namespace o atributo xsi:schemaLocation pode conter mais de um par de atributos[12].

Em XML Schema temos uma poderosa especificação para tipos. Dois grandes grupos de tipos estão disponíveis : tipos simples e tipos complexos. Os tipos simples constituem-se tipos de dados predefinidos baseados nos mesmos conceitos de tipos das linguagens de programação[12]. Os tipos de dados simples podem ser expandidos. No que se refere a tipos de dados complexos, existem mecanismos especificados que possibilitam a agregação de tipos para formar tipos compostos por elementos e outros tipos aninhados.

No que se refere a tipos de dados simples, XML Schema disponibiliza dois modos para a manipulação destes. No primeiro modo a especificação de XML Schema possui um vasto conjunto de tipos de dados básicos predefinidos entre os quais podemos citar e exemplificar (entre parênteses) os seguintes : string (confirme se isto é elétrico), base64Binary (GpM7); hexBinary (0FB7); integer (-12678); positiveInteger (1345); negativeInteger (-234); nonNegativeInteger (12678); nonPositiveInteger (-782); decimal (1.23); boolean (true,false); time (13:20:00.000-05:00); dateTime (1999-05-31T13:20:00.000-05:00); duration (P1Y2M3DT10H30M12.3S); date (1999-05-31); Name (shipTo); Qname (po:USAadress); anyURI (http://www.example.com); ID (especificação XML 1.0 ID attibute type); IDREF (especificação XML 1.0 IDREF attibute type)[12]. Para os tipos ID e IDREF temos a salientar que um processador XML pode causar os seguintes erros : para o caso de mais de um atributo com ID repetido e para o caso de IDREF fazer referência a um valor de ID que não pode ser encontrado[12].

A segunda forma com a qual XML Schema pode lidar com tipos de dados simples é permitindo a criação de novos tipos. O novo tipo simples a ser definido precisa ser derivado a partir de um tipo simples predefinido ou de um outro tipo simples criado anteriormente[12], chamamos a este tipo que oferece origem ao novo tipo simples de tipo base. O novo tipo simples é criado a partir do tipo base, sobre o qual são aplicadas restrições disponibilizadas como facets (características)[12]. Estas restrições permitem a configuração do novo tipo simples de modo a atender as necessidades e requisitos para o qual foi criado. O conjunto de restrições é composto por : length; minLength; maxLength (que restringem o tamanho em exato, menor e maior); pattern (possibilita uma expressão regular para o valor); enumeration (lista de todos os possíveis valores); whiteSpace (regras para manipulação de espaço em branco dentro do valor); minExclusive; minInclusive; maxExclusive; maxInclusive (faixa de valores numéricos permitidos); totalDigits (numero de dígitos decimais em valores numéricos); fractionDigits (numero de dígitos decimais após o ponto decimal)[12]. Estas características são aplicadas aos novos tipos simples a serem definidos respeitando, obviamente, a natureza do tipo base.

Os tipos complexos, em XML Schema, são constituídos a partir de mecanismos capazes de definir modelos complexos de conteúdo como por exemplo elementos que podem possuir atributos e outros elementos filhos aninhados. A definição de tipos complexos endereça a seqüência (estrutura) e a multiplicidade dos elementos filhos, bem como nomes associados aos atributos e a sua importância no que se refere presença obrigatória ou opcional[12].

Para construção de tipos complexos em XML Schema, emprega-se o elemento <xsd:complexType name=”nome_do_tipo_complexo> que identifica o início da definição de tipo complexo. Após o elemento inicial, deve seguir algum dos elementos disponíveis para a definição do modelo de grupo do tipo complexo que funcionam como agregadores de elementos filhos.

Entre as várias opções disponíveis para especificar o modelo de grupo de tipo complexo, as mais empregadas são : xsd:sequence (seqüência de elementos); xsd:choice (permite um entre o conjunto de elementos listados); xsd:all (permite um certo conjunto de elementos aparecerem um vez mas não todos, porém em qualquer ordem); xsd:group (referencia modelo de grupo definido em outro lugar)[12]. Após o elemento que especifica o modelo de grupo, pode-se adicionar qualquer quantidade de atributos através do elemento xsd:attribute[12].

O conceito da reusabilidade é fundamental para XML Schema[12]. Através desse conceito podemos gerenciar a questão de como adicionar o que de melhor já foi criado em projetos já em uso para ser empregado em novos projetos. No caso de XML Schema, podemos efetivamente

reutilizar definições de atributos e elementos, definições de conteúdo de modelos, tipos de dados simples e complexos e todo o XML Schema[12]. A reusabilidade pode ser dividida em dois níveis : básico e avançado. A reusabilidade básica emprega o mecanismo de solucionar problemas através do uso de definições e XML Schema já construídos, testados e aprovados em diferentes lugares e aplicações. No nível avançado os mecanismos de reusabilidade são empregados no sentido de resolver problemas através da modificação, da adaptação de definições e XML Schema já construídos para atender a requisitos e necessidades muitas vezes diferentes daquelas para as quais foram inicialmente projetados[12].

Temos no nível básico de reusabilidade as seguintes formas de empregar XML Schemas : referência a elementos, conteúdo de modelo de grupo, grupo de atributos, inclusão de XML Schema e importação de XML Schema[12].

Seguindo a especificação de XML Schema, podemos definir elementos usando tipo e nome. Alternativamente declarações de elementos podem se referir a elementos preexistentes utilizando o atributo xsd:ref, assim um elemento definido globalmente pode ser referenciado por outros elementos[12] de modo a evitar a redundância de definições. Estendendo o conceito de referenciamento de elementos, o conteúdo de modelo de grupo também pode ser referenciado no todo ou em partes. A forma como isto se dá é extremamente simples, através do atributo xsd:group que indica qual o modelo de grupo a ser associado[12]. Grupos de atributos podem encontrar aplicações em várias partes de um XML Schema, neste contexto de reusabilidade, o referenciamento destes grupos pode ocorrer de modo natural através do atributo xsd:attributeGroup[12].

Referâncias para elementos, modelos de grupos e grupos de atributos permitem a aplicação do conceito de reusabilidade apenas no mesmo XML Schema. Todos os referenciados (usados mais de uma vez) e os que referenciam (os que usam) devem estar no mesmo XML Schema. Entretanto quando se está lidando com XML Schemas muito complexos ou se deseja atingir o máximo de reusabilidade possível temos que, quase sempre, dividir o XML Schema em várias partes[12] para reduzir a complexidade e aumentar a capacidade humana de gerenciamento desta.

Mecanismos especificados e disponibilizados para importação e inclusão de XML Schemas em outros XML Schema, permitem e incentivam a divisão de grandes XML Schemas em outros menores, possibilitando que um documento referencie o outro[12]. A inclusão de um XML Schema em outro permite a utilização dos tipos definidos no XML Schema incluso pelo XML Schema que o está incluindo (recebendo) de modo normal. Para incluir um XML Schema em outro emprega-se

o atributo xsd:include, sendo que o XML Schema incluso passa a utilizar o mesmo namespace do XML Schema que o está incluindo, como que fazendo parte integrante deste[12].

O mecanismo de importação de um XML Schema por outro possui as mesmas funcionalidades do mecanismo de inclusão, porém diferencia-se no que se refere ao atributo a ser usado que passa a ser o xsd:import e no gerenciamento do namespace que neste caso mantém-se inalterado para o XML Schema importado[12] ou seja ao importar um XML Schema o XML Schema importador passa a contar com dois namespaces diferentes o seu, original, e o namespace do XML Schema importado.

Algumas aplicações de XML Schema requerem nível mais sofisticado de reusabilidade. De modo geral existem aplicações reais de XML Schema que necessitam de funcionalidades muito similares ao conceito de herança como definido em orientação a objetos. Para atender a este tipo de requisito, mais sofisticado, a definição XML Schema disponibiliza as extensões que devem ser usadas em conjunto com os tipos simples e complexos de dados[12] para simular o funcionamento de herança entre tipos, como o conhecemos da orientação a objetos [26] .

Temos a possibilidade de estender e restringir tipos de dados simples e complexos através dos atributos xsd:extensions e xsd:restrictions respectivamente. Quando desejamos expandir ou restringir tipos de dados simples tomamos como base um tipo de dado predefinido e sobre este aplicamos a restrição ou a extensão que deve ser expressa em termos de um facets adequado ao tipo. Nesta operação emprega-se o elemento xsd:simpleType[12]. Para o caso de tipos complexos utiliza-se geralmente como tipo base outro tipo complexo do qual se deseja estender o conteúdo complexo adicionando-se elementos e outros mecanismos ou deseja-se restringir o conteúdo, descartando-se componentes da estrutura original[12]. Com estes recursos a aplicação de XML Schema possui todo o poder, a precisão e a flexibilidade para a definição de esquemas de validação de documentos XML, aspecto fundamental para a implementação de Web Services e demais tecnologias relacionadas.

3.3 - SOAP

Em arquiteturas de Web Services, o protocolo básico padrão de fato para comunicação entre duas partes é o SOAP (Simple Object Acess Protocol)[11]. Estabelecendo-se como protocolo básico de mensagem para Web Services, outras especificações que agregam facilidades e estendem possibilidades de uso do SOAP são construídas e definidas sobre o mesmo, também ocorre de haver

definições que acoplam o SOAP sobre algumas camadas de protocolos e meios de transporte de mensagens.

SOAP constitui-se protocolo baseado em XML. A principal inovação introduzida por SOAP na computação distribuída é a possibilidade de comunicação efetiva entre sistemas distribuídos estabelecidos sobre conjuntos de software e hardware heterogêneos[11]. SOAP foi projetado em um contexto onde as possibilidades para sistemas distribuídos caracterizavam-se por soluções baseadas em forte acoplamento entre as partes.

Uma das características mais importantes do SOAP é a separação do formato do dado a ser transmitido, do protocolo de nível inferior que irá transportar o dado, produzindo independência de plataforma e linguagem, pois emprega-se XML para descrever os dados e utiliza-se protocolos bem estabelecidos, como HTTP, para transportar dados através da Internet[11].

Alguns aspectos colaboram para a simplicidade do SOAP, entre estes podemos citar o fato de que SOAP não precisa manter compatibilidade com nenhum protocolo antecessor[11], por exemplo SOAP não precisa suportar coleta de lixo distribuída, não precisa usar nenhum software de broker e nem suportar outras funcionalidades de protocolos anterios como JRMP, IIOP e outros.

Inicialmente SOAP foi concebido para funcionar sobre HTTP. Em sua natureza de projeto HTTP constitui-se protocolo extremamente simples, um cliente faz uma solicitação empregando GET/POST com algum dado e recebe como resposta um arquivo (geralmente em HTML) ou uma mensagem de erro[11]. O HTTP não cria restrições para o dado, o modo como o cliente processa o dado ou a infra-estrutura de rede que opera sob ele. Tudo que o HTTP necessita é estabelecer uma conexão TCP/IP entre dois pontos[11] e transportar uma solicitação e uma resposta a esta solicitação.

A indústria de computação realizou movimento tecnológico para definir padrão XML aderente ao modelo de computação distribuída baseado em RPC. SOAP representa a evolução deste movimento definindo o padrão XML baseado em RPC sobre o protocolo HTTP, proporcionando assim extrema simplicidade[11]. Inicialmente desenvolvido por empresa denominada Userland Software em meados de 1998. Tendo, em 1999, a adesão da Microsoft e da empresa Develop Mentor que realizando esforço em conjunto publicaram a especificação SOAP 0.9 a qual não foi imediatamente aceita pelo mercado. Como evolução da versão anterior, a especificação para SOAP 1.1 que incluía a possibilidade de transportar anexos foi rapidamente aceita e padronizada como protocolo de fato para Web Services[11].

A especificação do protocolo SOAP limita-se a formalizar questões básicas que informam precisamente como aplicações diferentes, estabelecidas em ambientes de comunicação distintos (heterogêneos ou não) devem proceder para comunicar-se[11] ou seja trocar mensagens SOAP. Como efeito positivo da simplicidade estabelecida com SOAP, vários consórcios na industria trabalham no desenvolvimentos de padrões para os vários aspectos considerados essenciais em sistemas distribuídos, entre estes podemos citar segurança, transações e padrões de mensagens[11]. SOAP tem a capacidade de expandir-se para acomodar tecnologias e padrões emergentes, bem como acomodar padrões já estabelecidos[11].

Projetado para manter a compatibilidade, capacidade de expansão e adequação ao ambiente de rede e também de Internet, SOAP pode expandir-se em dois sentidos : para baixo, tornando-se aderente a padrões de protocolos de comunicação e para cima, permitindo a adição de serviços que agreguem valor, em camadas superiores.

A expansão para baixo de SOAP está concretizada no fato de que o mesmo é, por concepção, independente do protocolo de transporte. Não há especificação alguma que imponha como requisito para transporte de qualquer parte de uma mensagem SOAP, a necessidade explícita ou nominal de protocolo específico[11]. Assim, podemos enviar mensagens SOAP empregando HTTP, correio eletrônico (SMTP/POP), rwa socktes, arquivos de texto e até mesmo a empresa de correios[11].

O crescimento expansivo de SOAP para cima ocorre da capacidade de acomodar, em camada superior, a adição de classes robustas e altamente confiáveis para manipular filas de mensagens, transações, mensagens seguras e outras características necessárias a aplicações do mundo empresarial, a partir de mensagens básicas do protocolo SOAP[11]. Muitas especificações como por exemplo de segurança e transações, podem ser adicionas como funções de valor agregado ou protocolos de camada superior[11].

SOAP acomoda estas adições definindo modelo de processamento para mensagens no qual os nós intermediários do SOAP, processam as mensagens baseados nas instruções contidas nos header da própria mensagem SOAP. Muitas destas extensões são resultado das vantagens adicionadas pela especificação de SOAP com Anexos[11]. A Figura 3.3-1 a seguir demonstra como SOAP pode crescer tanto para cima como para baixo e relacionar-se com as camadas de transporte e expansão dos serviços, comportando-se como o catalisador entre o nível de rede e as tecnologias, emergentes ou não, necessárias a computação distribuída.

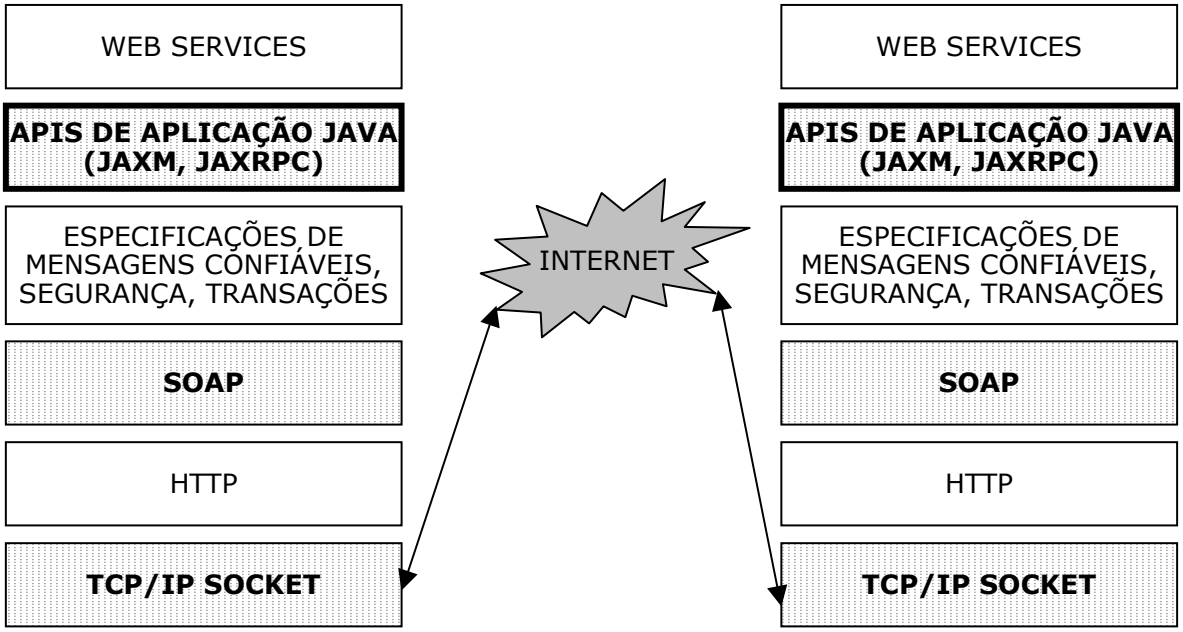


Fig. 3.3.1-1 pilha de comunicação Web Services

3.3.1 – SOAP : aspectos fundamentais

O protocolo SOAP restringe-se em definir características básicas da informação que podem ser resumidas em três aspectos : o modo como a mensagem XML é estruturada; convenções que representam a chamada remota de procedure (RPC) na mensagem XML; a comunicação como o protocolo HTTP para garantir que a mensagem XML seja transportada corretamente e convenções para representar a ocorrência erros que devem ser informados de volta ao emissor da mensagem SOAP[11].

As definições do protocolo SOAP não referem-se a modelo de objeto ou a linguagens de ligação com outros componentes de tecnologia[11], focam, tão somente, em um modo formal para que mensagens XML possam ser comunicadas entre um emissor e um receptor, genericamente denominados nós de processamento SOAP, que podem comportar-se como emissores, receptores ou ambos[11].

A definição sentido único da mesnagem no sentido do emissor para o receptor, é sempre combinada com outros recursos para que se implemente outros modelos de interação de mensagens e comunicação como solicitação-resposta, mensagens assíncronas e notificações[11]. No caso específico de HTTP um cliente envia uma solicitação com uma mensagem SOAP, o servidor processa a solicitação e responde com outra mensagem SOAP. Temos assim a adaptação do protocolo

SOAP sobre o protocolo solicitação-resposta HTTP. A Figura 3.3.1-2 apresenta os possíveis comportamentos dos nós de processamento SOAP.

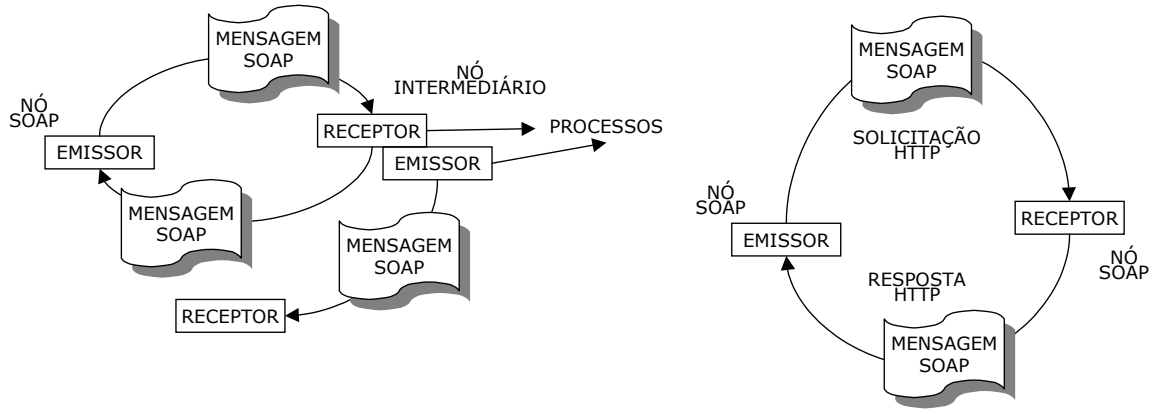


Fig. 3.3.1-2 – nós de processamento SOAP

3.3.2 – Estrutura da mensagem SOAP

Uma mensagem SOAP constitui-se em documento XML bem formado que contém elemento raiz denominado Envelope, com elementos opcionais Header e um elemento obrigatório Body[11].

O elemento Envelope funciona como container para os elementos Body e Header e como um indicador de o documento XML constitui-se em uma mensagem SOAP. A principal função do elemento Envelope é indicar ao receptor da mensagem aonde começa e onde termina a mensagem. Ao encontrar o marcador </Envelope> do elemento Envelope o receptor sabe que pode começar a processar os anexos da mensagem, se houverem. Essencialmente o Envelope constitui-se uma estrutura destinada ao empacotamento de mensagens SOAP[11].

Opcionalmente uma mensagem SOAP pode conter o elemento Header. Porém caso esteja presente, o elemento Header deve ser o primeiro imediatamente após o elemento Envelope. Dentro do elemento Header pode haver qualquer quantidade de entradas, são livres, pois a especificação SOAP não define nenhuma destas entradas em particular ou qualquer bloco XML que venha a fazer parte do elemento Header. Esta é uma das possibilidades de extensão do SOAP que é explorada por outras especificações que o fazem através da padronização de entradas no elemento Header[11]. Isto significa que ao adicionar alguma funcionalidade, a nova especificação pode definir entradas no elemento Header denominadas headers. As mensagens SOAP carregarão headers que fazem parte de uma especificação para serem processados de modo padrão pelos nós que os

reconhecerem e estiverem preparados para tal[11]. A utilização de headers pode ser muito variada, podemos ter headers que contenham metadados auxiliares para descrever ou aumentar o conteúdo principal da mensagem SOAP, header com informação de roteamento da mensagem por diferentes rotas, header adicionais de segurança e outros[11].

Todas as mensagens SOAP tem que conter obrigatoriamente o elementto Body. Este elemento carrega conteúdo para o recipiente final da mensagem. Este conteúdo pode ser composto por elementos XML que descrevem a chamada a uma procedure definindo parâmetros e argumentos juntamente com o nome da procedure a ser chamada ou outro tipo de conteúdo XML como um documento nota fiscal de venda. Para estas duas possibilidades de conteúdos XML que o elemento Body pode transportar existem nomes genéricos : mensagem SOAP estilo RPC e mensagem SOAP estilo documento, respectivamente[11]. O elemento Body pode conter, entretanto, qualquer tipo de mensagem XML.

As regras que permitem que uma mensagem SOAP seja transportada por um protocolo específico são denominadas SOAP binding (ligações SOAP)[11], estas regras atuam de modo a permitir que uma mensagem SOAP seja transmitida para uma camada inferior responsável em fazer o transporte desta mensagem até o destino. Por outro lado, os headers de uma mensagem SOAP determinam como um nó de processamento SOAP deve manipular uma mensagem e o que deve ser realizado com a mesma[11]. A Figura 3.3.2-3 demonstra como as três principais partes de uma mensagem SOAP estão estruturadas bem como ocorre o relacionamento da mensagem com o protocolo HTTP.

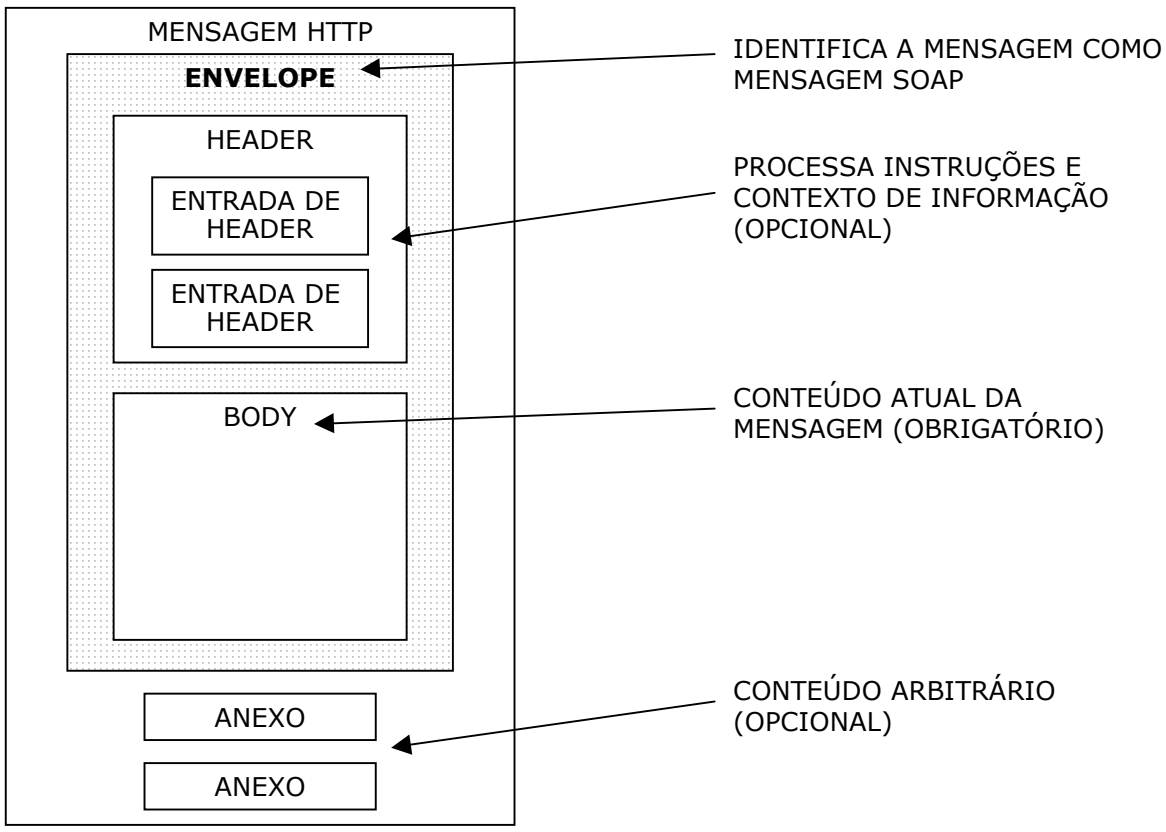


Fig. 3.3.2-3 – estrutura da mensagem SOAP

3.3.3 – Elementos da mensagem SOAP

Mensagens transmitidas através do protocolo SOAP são constituídas por três elementos distintos : Envelope, Header e Body. Entre estes, o elemento Header é opcional. Além destes elementos, o protocolo SOAP define mecanismos de manipulação de erros que devem ser reportados ao emissor da mensagem. Para implementar esta função o protocolo SOAP define ainda o elemento Fault.

Principal e mais externo container de mensagem do protocolo SOAP constitui-se do elemento Envelope. Este elemento identifica para o nó de processamento que a mensagem trata-se de uma mensagem do protocolo SOAP[11], esta é a sua principal função, delimitar o início e fim da mensagem SOAP. O recurso XML namespace é empregado neste elemento, no lugar do número de versão, para informar ao nó de processamento SOAP a versão do protocolo que esta sendo usada no envelope corrente[11]. Diferindo de outros protocolos como HTTP que trazem a versão em local fixo, a versão do envelope de SOAP para o envelope corrente pode ser inferida. As mensagens SOAP são pesadamente baseadas no mecanismo XML de namespaces, isto ocorre porque uma mensagem pode conter qualquer tipo de elemento XML, inclusive com o mesmo nome dos elementos padrão do protocolo SOAP, para evitar possíveis conflitos, como indicado em [12], os elementos inerentes ao

protocolo SOAP devem ter seu escopo definidos de alguma forma[11]. Um nó SOAP pode suportar, processar uma ou mais versões de SOAP simultaneamente. Entretanto, se um nó de processamento SOAP recebe versão que não possa processar a especificação SOAP orienta que esta mensagem deve ser descartada e o nó SOAP deve enviar ao emissor, em um cenário solicitação/resposta, mensagem de falha sinalizando "version Mismactch" (versão inadequada)[11].

O elemento SOAP Header possibilita um container, uma seção a qual podem ser adicionados metadados relativos ao conteúdo do elemento SOAP Body e/ou instruções para processamento específico da mensagem SOAP[11]. Header são elementos opcionais mas constituem-se no mecanismo chave para a extensão vertical do protocolo SOAP[11], isto ocorre pela adição apropriada de entradas no Header que instruem nós de processamento a comportarem-se de modo específico em determinados contextos como : autenticação, autorização, transições e roteamento entre outros[11]. Em alguns cenários, informações incluídas no Header estão relacionadas com características críticas de processamento da mensagem, nestes casos SOAP define para estas entradas o atributo mustUnderstand (entrada no header deve ser processada corretamente). Ao receber mensagem SOAP com este atributo, o nó deverá executar o que determina a entrada ou, em caso de falha, enviar mensagem padrão, bem definida ao emissor da mesma[11].

A mensagem transmitida do provedor para o requisitante ou vice-versa, via protocolo SOAP, vai contida no elemento Body[11]. Existem dois estilos possíveis de mensagens : estilo RPC, e o estilo documento XML completo. No estilo RPC, são descritos detalhes sobre a procedure a ser chamada e os parâmetros adequados, empregando-se modo de codificação[11] negociado entre as partes que se comunicam. O estilo documento XML completo, como o próprio nome informa, deve conter um documento XML que está sendo trocado entre as partes[11]. Para ambos os casos a mensagem SOAP carregada pelo Body deve ser bem formada e válida entre o emissor e o receptor, a única diferença que pode haver é o modo como o recipiente emprega o esquema de codificação para caracteres[11].

SOAP define mecanismo sofisticado e preciso para informar ao emissor a ocorrência de algum erro ou falha no processamento de mensagens. Embutindo o elemento XML padrão e bem formato denominado Fault no Body, o nó SOAP indica que ocorreu algum problema no processamento da mensagem recebida[11], este tipo de mensagem é genericamente conhecida como falha SOAP. Podem ocorrer falhas ocasionadas por grande número de razões como pela aplicação, pelo sistema e outros componentes do contexto.

Somente o elemento simples Fault deve fazer parte da mensagem de falha especificada pelo SOAP. Este elemento pode conter os seguintes subelementos : o faultcode, obrigatório, indicando a classe geral dos erros; faultstring, obrigatório, contendo uma descrição por extenso do erro ocorrido; faultactor, que indica a fonte URI onde ocorreu o erro, este elemento é obrigatório se o erro ocorrer em um dos intermediários e opcional se o erro ocorrer no recipiente final e detail elemento opcional, empregado para informar detalhes de processamento do erro[11]. Se a falha for provocada por erros no processamento do Header este elemento não deve estar presente, entretanto se a falha for provocada pelo processamento de solicitações presentes no elemento Body este elemento deve indicar detalhes desta falha podendo conter sub elementos que devem ter seu conteúdo ajustado ao esquema de codificação de caracteres[11].

Para comunicar falhas e erros ocorridos, o elemento faultcode possui mecanismos extensíveis, através de nomes qualificados em XML. O ponto (".") é empregado como separador para indicar que o código mais a esquerda é o faultcode mais genérico. SOAP define as seguintes classes gerais de falhas para o elemento faultcode : versionMismatch (versão inadequada) indicando que o nó SOAP não pode processar a versão do SOAP na mensagem recebida; mustUnderstand (entrada no header deve ser processada corretamente) quando uma entrada obrigatória no Header não puder ser processada corretamente; client quando o erro ocorrer no cliente, significando que o mesmo informou algo de modo errado ou insuficiente ou ainda que a solicitação enviada pelo cliente não foi formulada corretamente, de um modo genérico, este erro indica que a origem do problema está no cliente cliente; server indica que o erro ocorreu no servidor, erros ocorridos no processamento de aplicações e componentes são informados nesta classe[11].

3.3.4 – Modelo de processamento SOAP

A mecânica pela qual um nó manuseia mensagens SOAP é denominada de modelo de processamento[11] ou seja o comportamento de um nó ao enviar, receber ou ambos com relação a mensagens SOAP. Além do emissor e do receptor, na especificação SOAP, temos ainda os nós que estão entre ambos e realizam a tarefa de receber e enviar a mensagem. Esta característica possibilita a extensibilidade horizontal do protocolo SOAP. Um intermediário pode receber uma mensagem do emissor ou de outro intermediário e transmiti-la para o destinatário ou para outro intermediário[11]. Assim temos em SOAP três categorias de nós de processamento : emissor, receptor e intermediário.

Uma mensagem SOAP pode destinar headers para nós específicos, de modo a garantir que a mensagem seja roteada passo a passo, obedecendo a seqüência de processamento

preestabelecida[11]. O SOAP para o qual a mensagem no Header foi destinada pode ser o recipiente final (o receptor) ou qualquer um dos nós intermediários. Cada entrada no header deve conter o nome do nó a qual se destina, esta identificação é informada através do atributo actor na entrada do header[11].

O modelo para identificar o actor (nó de processamento SOAP) que deve agir sobre o header da mensagem é baseado em duas premissas : todos os nós SOAP são identificados por uma URI única e uma URI pode ocorrer em um ou mais headers,[11] fazendo com que o mesmo nó atue sobre um ou mais headers da mesma mensagem. A URI, neste caso, funciona apenas como critério lógico de identificação única[11], não precisa existir fisicamente. SOAP define dois tipos especiais para o conteúdo do atributo actor : o tipo especial padrão e constante de URI contendo http://schemas.xmlsoap.org/soap/actor/next indicando que o bloco do header não deve ser processado pelo nó que recebeu a mensagem. E no caso do atributo actor não ser especificado, o nó assumirá que a entrada no header, não pertence a ele, devendo ser processada pelo receptor final da mensagem, o destino[11]. Estes dois casos são previstos por serem casos comuns e freqüentes de uso e torna-se prático a especificação de detalhes como estes, independentemente de por onde a mensagem venha trafegar ou como ela seja enviada entre as partes[11]. Esta facilidade evita que a estrutura de controle da mensagem seja alterada caso o meio em que trafegue sofra algum tipo de mudança.

Além do emissor, nós intermediários do protocolo SOAP também podem inserir entradas no elemento Header. O ato de inserir uma entrada header em uma mensagem tem o mesmo significado que estabelecer um contrato com o emissor e com o receptor da mesma. Um intermediário que insere uma entrada no header de uma mensagem comporta-se de forma similar ao nó que originou a mensagem, faz papel semelhante ao emissor. Entretanto o receptor enxerga a mensagem como se fosse enviada pelo intermediário e não pelo emissor[11]. Ao intervir em uma mensagem o intermediário deve tomar conhecimento e manipular somente os blocos de header endereçados a ele, porém não existem garantias de que o intermediário não possa verificar outros blocos, além dos seus[11].

Os atributos action e mustUnderstand podem ser combinados para possibilitar roteamento da mensagem por trajetos compostos de vários nós de processamento previamente selecionados[11]. Este modelo de processamento distribuído de mensagem através de nós possui uma arquitetura altamente escalonável, na qual cada nó pode agregar valor e funcionalidades para atender as solicitações[11].

De modo genérico um nó SOAP segue quatro passos básicos para o processamento de uma mensagem : verifica se uma mensagem é SOAP; identifica e processa as entradas do header

destinadas a ele, deve entender quando o header tem processamento obrigatório e em caso de falha deve retornar mensagem de falha padrão SOAP Fault; se o nó for um intermediário deve enviar a mensagem adiante e se for o nó destino deve processar o conteúdo do elemento Body. SOAP define este modelo de processamento empregando apenas os conceito de atores e headers[11] no caso atores são os nós de processamento. SOAP não define nenhuma mensagem para protocolo de roteamento[11]. A Figura 3.3.4 – 4 a seguir oferece representação simples do modelo de processamento demonstrando o comportamento dos nós emissor, receptor e intermediários.

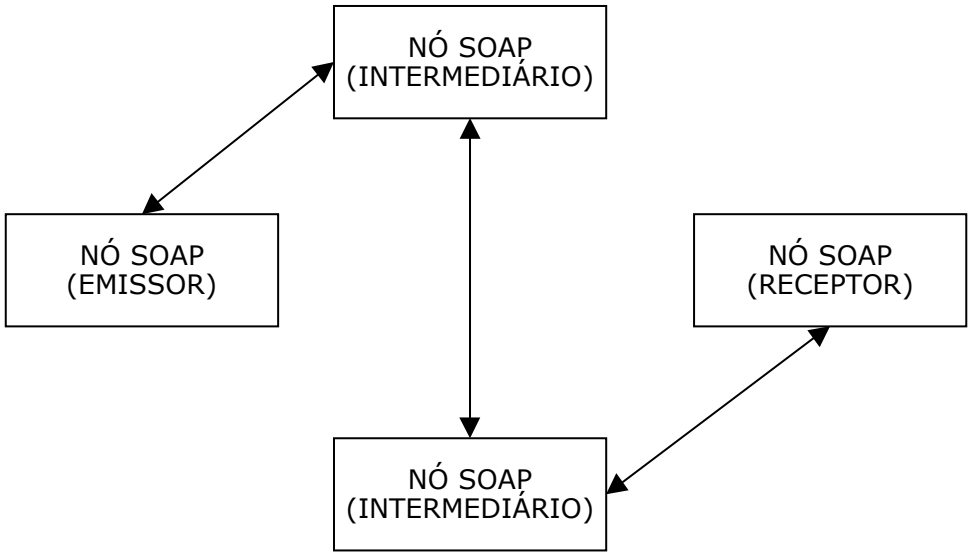


Fig. 3.3.4-4 nós SOAP e intermediários

3.3.5 – SOAP sobre protocolos de transporte

Constituindo-se em protocolo de aplicação, uma mensagem SOAP possui, essencialmente, a capacidade de enviar comunicação em apenas um sentido[11]. Mecanismos de ligação que definem como mensagens SOAP devem ser processadas entre os nós, constituem-se um dos modos pelos quais podemos estender as funcionalidades básicas do SOAP a partir das camadas inferiores dos protocolos de transporte, os quais já possuem muitas das funcionalidades necessárias[11]. Em alguns casos (como mensagens confiáveis e mensagens seguras) são necessárias especificações adicionais. Em outros casos basta que as mensagens SOAP sejam passadas para as camadas a baixo onde padrões já estabelecidos executam a funcionalidade requerida[11]. A versão SOAP 1.1 disponibiliza especificações para ligações com HTTP e define como o modelo solicitação-resposta de comunicação deve comportar-se sobre HTTP[11]. A utilização de SOAP sobre HTTPS também satisfaz algumas das necessidades de aplicações reais no que diz respeito aos requisitos de segurança[11].

A definição SOAP especifica como ligar a transmissão de mensagens SOAP sobre o protocolo HTTP POST no mecanismo de solicitação[11]. Esta definição formal direcionada para HTTP tira proveito de funções anteriormente implementadas no HTTP para facilitar a manipulação de mensagens SOAP[11]. Desta forma, nós HTTP com capacidade para processar mensagens SOAP são acrescentados por regras que indicam como processar uniformemente estas mensagens empacotadas em solicitações padrão do próprio HTTP[11]. Entre outras possibilidades de aderência, nós SOAP sobre HTTP podem tirar proveito do código HTTP 500, como resposta, quando alguma falha ocorrer[11].

O header do HTTP denominado SOAPAction indica o tipo de ação ou destinação da mensagem SOAP empacotada[11]. Quando trata-se de mensagem estilo RPC, nenhum valor é informado para este header, o que deve ser feito com a mensagem é inferido de outras formas mais sofisticadas dentro do próprio protocolo SOAP, que decidirá para onde a mensagem será enviada, segunda a indicação de alguma URI presente na mesma ou que tipo de processamento será empregado[11].

No caso de mensagens estilo documento o significado dos dados e o seu destino serão inferidos após o processamento de todos os dados, o que torna o processamento lento e contraria as boas práticas que pregam a separação entre dados e metadados[11]. Para evitar todos estes inconvenientes e considerando o cenário da transmissão no estilo documentos, o campo HTTP deve ser usado para indicar a destinação da mensagem, no que se refere a solicitação recebida[11]. Como trata-se de um campo do HTTP, em nível inferior a camada SOAP o mesmo pode ser interpretado sofrendo o parse de forma relativamente rápida[11]. Este aspecto satisfaz os princípios das boas práticas [49] separando os metadados dos dados que estão sendo enviados no Body do SOAP[11]. Outra aplicação possível para o campo SOAPAction é para a informação a firewalls no sentido de que pacotes com este campo, sejam filtrados e rapidamente enviados evitando o congestionamento de tráfego[11].

Ao utilizar a parte de resposta nativa do HTTP, Web Services baseados em SOAP podem implementar o padrão de troca de mensagens denominado solicitação-resposta, mesmo que este modelo não esteja definido no protocolo SOAP[11]. Estendendo esta possibilidade, poderemos adaptar outros mecanismos HTTP como PUT para implementar o padrão de troca de mensagem sentido único com confirmação de recebimento[11]. Além deste avanço poderemos ainda desenvolver a resposta HTTP 2XX para indicar que o processamento foi bem sucedido, no caso de falhas definidas da aplicação o elemento SOAP Fault poderá ser usado para indicar o erro ocorrido e para falhas nas quais

a solicitação não puder ser processada o nó SOAP deverá retornar o código HTTP 500 para erro interno do servidor[11].

Em cenários que exijam o modelo padrão de troca de mensagem solicitação-resposta, SOAP sobre HTTP atende as necessidades. Porém existem casos em que o modelo solicitação-resposta demonstra-se inadequado. Assim, utilizando protocolos de correio eletrônico como SMTP e POP3, aplicações e Web Services baseados em SOAP, podem tirar vantagem dos modelos padrões para trocas de mensagens assíncrono e armazena e envia como camadas de transporte, em sentido único, para o protocolo SOAP[11]. Esta possibilidade permite o emprego do SOAP em um número muito maior de cenários os quais somente o protocolo HTTP não poderia atender adequadamente[11]. Como a especificação SOAP 1.1 não menciona como o protocolo SOAP e seus anexos devem ser tratados por protocolos de transporte como SMTP e POP3, torna-se simples inferir como isto deve ser feito[11].

3.3.6 – Codificação de caracteres no protocolo SOAP

Em qualquer protocolo de computação distribuída, os estados de objetos e aplicações serão freqüentemente trocados entre os nós participantes[11]. Para que isto ocorra é necessário que se estabeleça acordo entre todos os participantes sobre qual protocolo será usado para que o estado dos objetos e aplicações possa ser transmitido entre eles. No caso de Web Services isto significa que mensagens SOAP, as quais carregam a representação em XML do estado de objetos, devem ser representadas em uma forma padronizada de modo que todas as partes envolvidas (nós de processamento SOAP) possam entender e interpretar a informação sobre os estados dos objetos[11]. A codificação de caracteres refere-se as regras e normas de codificação antes da transmissão (serialização) e descodificação após a recepção (deserialização) especificamente de objetos transportados por mensagens SOAP. Esta codificação pode ser denominada na literatura como "Section 5" devido sua definição encontrar-se na seção cinco da especificação SOAP[11].

Web Services tem como uma de suas principais características o fato de poderem ser implementados em um grande número de linguagens de programação. As estruturas de dados definidas nestas linguagens de aplicação devem ser representadas em XML para a transmissão[11]. Esta possibilidade é derivada a partir da ligação, da aderência da linguagem ao padrão XML, neste caso estamos falando de exemplos como Java para XML e XML para Java, C++ para XML e XML para C++, para citar apenas alguns[11]. Entretanto, a partir do ponto em que um dado pode ser satisfatoriamente representado em XML, a mensagem SOAP pode transporta-lo em seu elemento Body de dois modos : baseado em acordo entre as partes sobre um esquema padrão que define os conteúdos

do XML, geralmente é denominado de codificação literal, onde os tipos não são específicos de nenhuma linguagem e estão no conjunto de tipos primitivos da especificação XML Schema[11] . A outra forma é através de regras predeterminadas estabelecidas sobre esquema comum, aceito por todas as partes envolvidas, no qual se descreve a codificação do conteúdo XML a ser trocado[11]. Esta modalidade é completamente opcional, segunda a especificação SOAP que oferece a Seção 5 somente como conveniência a fim de permitir que os nós troquem informações sem que haja a necessidade de conhecimento prévio sobre a mesma[11].

Em ambos os casos, o emissor e o receptor devem empregar o mesmo esquema de codificação antes da transmissão (serialização) e descodificação após a recepção (deserialização) para obterem o correto processamento da mensagem[11]. Ambos devem concordar sobre os esquemas no que diz respeito a : emprega-lo sobre todas as mensagens SOAP sem exceção, usar o mesmo mecanismo de codificação, entenderem os mesmos headers da mesma forma e documentos XML específicos da aplicação no elemento Body e seus anexos. Mesmo que este item não seja necessário, constitui-se uma boa prática acordá-lo para conteúdos XML[11].

SOAP define esquema de codificação obrigatório para todas as mensagens. O valor do atributo encodingStyle, que deve ser uma URI, indica ao nó SOAP que está recebendo a mensagem o apontador para as regras a serem empregadas na codificação e na decodificação da mensagem recebida[11]. O conteúdo padrão http://schemas.xmlsoap.org/soap/encoding é empregado para indicar que o conteúdo do elemento Body e do elemento Header devem ser codificados pelo padrão determinado pelo documento e não por padrões individuais de cada um, se existirem[11].

A especificação SOAP define, através de esquemas e com certo rigor, conjunto de regras para a codificação que mapeiam bem formadas construções de programação, entretanto não há a definição de nenhum esquema de codificação padrão[11] que deve ser assumido caso o encodingStyle não seja informado ou encodingStyle seja igual a " ". Isto significa que se encondingStyle não aparecer ou aparecer com encodingStyle=" ", o receptor não poderá assumir nenhuma codificação como padrão sobre como os dados devem ser representados na mensagem e terá que tentar inferir por si mesmo como decodificar a informação[11].

Para a representação de tipos SOAP define dois grupos principais : tipos simples e tipos compostos. Os tipos simples são os mesmos tipos simples do XML que são compostos por dados simples dentro do corpo de um elemento[11]. A definição SOAP expôs para uso, como tipos simples, todos os tipos definidos na especificação XML Schema, adicionando duas alternativas de sintaxe para expressar as instancias destes tipos de dados[11], permitindo que os mesmos sejam prefixados de modo

normal ou ainda por vários namespaces. A informação sobre a codificação de tipos simples pode ser associada ao elemento de dois modos : através do prefixo xsi:type=" (tipo do elemento)" ou apontando diretamente para o esquema de codificação particular no próprio elemento com o prefixo xsi:schemaLocation[11]. Mesmo dispondo de cerca de três dúzias de tipos, os mais freqüentemente empregados são : string; integer; byte; int; long; decimal; float; double; Boolean; date e base64Binary (para representar conteúdo binário)[11].

Tipos compostos constituem-se na segunda maneira pela qual SOAP define a representação de tipos. Um tipo composto representa a junção, a união de dois ou mais tipos simples agrupados sob um elemento[11]. Além de agrupar tipos simples, os tipos compostos podem representar estruturas ou vetores (arrays). Uma Estrutura é um elemento composto por elementos filhos. Estruturas compostas empregam o mesmo mecanismo do a atributo xsi:type para indicar o esquema de codifição individual dos elementos[11].

Um vetor constitui-se em um elemento composto que contem elementos do mesmo tipo que se repetem. Vetores são codificados como elementos do tipo enc:type e possuem o atributo adiciona enc:arrayType para descrever o tipo do seu conteúdo, esta declaração toma a forma tipo_do_vetor[quantidade de elementos do vetor] que é similar a declaração de um vetor em uma linguagem de programação como Java[11]. Podemos definir um vetor composto por tipos simples e tipos compostos[11]. O suporte a vetores multidimensionais é fornecido através do uso de referências, similares as empregadas em HTML, que são especificadas pelo atributo href o qual aponta para o elemento identificado pelo atributo id[11]. Uma simulação (encadeamento hierárquico) que fornece a impressão de várias dimensões do mesmo array.

Apesar da flexibilidade oferecida pelo SOAP para que cada elemento possua seu próprio esquema de codificação, a prática tem demonstrado que apenas um mesmo esquema de codificação para toda a mensagem SOAP, designado pelo atributo encodingStyle é o mais recomendável e enquadra-se perfeitamente nos princípios de boas práticas da programação[11].

3.4 – WSDL

A pilha de tecnologia relacionada a Web Services, introduzida em [12], denominada Pilha de Descrição de Serviço fornece noção da camada destinada a descrição e composição de serviços. A tecnologia WSDL (Web Service Description Language) tem a função de descrever Web

Services de modo a preencher lacunas principais como: informar ao consumidor o que o serviço executa, como o consumidor pode invocar o serviço e como o consumidor pode diferenciar entre serviços similares oferecidos por diferentes fornecedores [11].

O W3C oferece a seguinte definição para WSDL : "WSDL constitui-se em formato XML para descrição de serviços de rede como conjunto de pontos de acesso, disponibilizando mensagens com informações que podem ter conteúdo orientado a documento ou conteúdo orientado a procedure. Operações para se obter as mensagens e as mensagens em si, são descritas de forma abstrata e ligadas concretamente ao protocolo de rede e ao formato da mensagem para definir o ponto de acesso. Relacionando pontos de acesso concreto com pontos de acesso abstratos (serviços). WSDL é extensível para permitir a descrição de pontos de acesso e suas mensagens através de formatos flexíveis para a mensagem e do protocolo de rede usado para comunicar, entretanto, somente as ligações descritas neste documento indicam como empregar WSDL em conjunto com SOAP 1.1, HTTP GET/POST e MIME."

A função principal do WSDL é permitir que o cliente interessado em utilizar o Web Service, possa faze-lo de forma automática, sem que seja necessário intervenção ou contato com o autor do mesmo.

3.4.1 – Descrição de Web Services

Para que possa ser plenamente empregado, Web Services devem ser descritos em três aspectos a considerar : como invocar o Web Service, que serviço oferece melhor qualidade quando vários provedores oferecem serviços com características similares e em que ordem serviços relacionados e suas operações devem ser invocados [11]. No âmbito mais básico da descrição a informação de como Web Services deve ser invocado deve esmiuçar detalhes como a interface, quais métodos estão disponíveis, as assinaturas do métodos e os valores retornados. Completam estas informações o endereço onde o serviço está localizado e o protocolo que o serviço é capaz de processar para realizar comunicação [11].

No aspecto de descrição do serviço relativo a qualidade inerente ao serviço a ser prestado, a descrição deve conter informações sobre o nível de segurança oferecido em relação a segurança dos concorrentes, garantias particulares de rapidez na resposta, maior escalabilidade e disponibilidade e finalmente, informações relativas ao acordo que deve ser firmado entre as partes, para

que o serviço possa ser utilizado, bem como, incluindo-se aí a política de tarifação pelo uso do mesmo[11].

Em nível mais avançado a descrição completa de Web Services poderá incluir a ordem como este e seus métodos devem ser invocados a fim de se poder estabelecer uma combinação de Web Services para criar um macro Web Services, também chamado de Orquestração de Web Services [11].

Estas três formas de descrição cobrem completamente todos os aspectos de Web Services que um potencial cliente poderia solicitar [11]. Entretanto um consumidor de Web Services não solicitarás as três descrições todas as vezes que utilizar o Web Service [11]. Neste contexto termos a considerar que um cliente invocando um Web Service isoladamente não estará interessado em formas de como combiná-lo com outros Web Services para realizar uma orquestração se serviços [11]. Por outro lado, no caso de haver apenas um fornecedor para um Web Service, características não funcionais como tempo de resposta, escalabilidade e outras, não serão importantes, neste cenário o principal aspecto é a utilidade do Web Service para o objetivo do cliente [11].

3.4.2 – Descrição de características funcionais de serviços

As características funcionais de um Web Service são pertinentes a interface do serviço (o como) mais informações de publicação, de disponibilização (o onde) que juntas atendem os requisitos necessários para que o mesmo possa ser invocado [11]. A arquitetura orientada a serviço não constitui-se em conceito novo, no passado serviços publicados tinham suas características funcionais descritas e divulgadas empregando diferentes mecanismos e padrões [11]. WSDL tem a função de divulgar informações detalhadas que descrevem como o cliente pode utilizar o Web Service, estas informações são chamadas de características funcionais. Assim, o serviço de descrição WSDL constitui-se uma gramática e vocabulário XML para descrever Web Services [11].

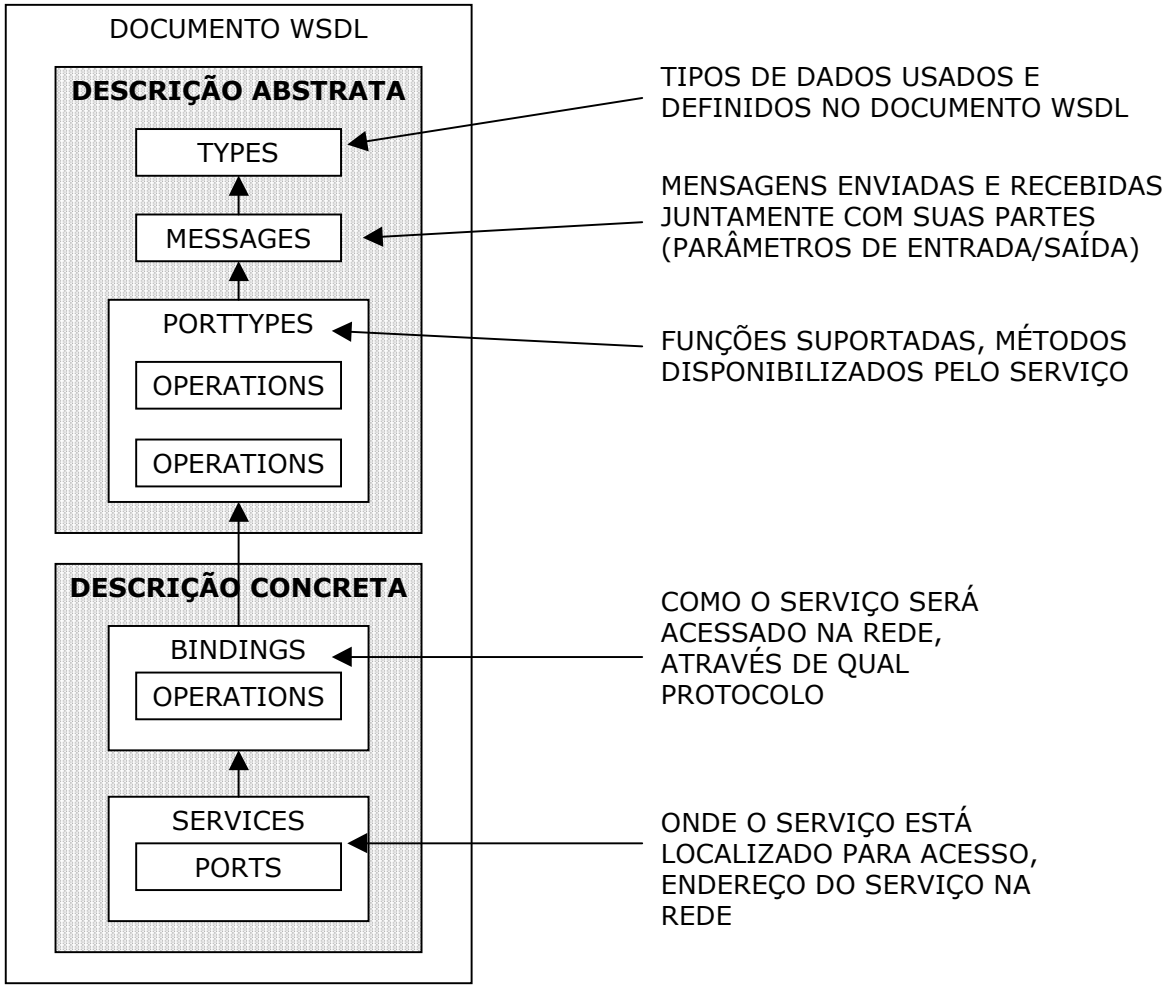
3.4.3 – Fundamentos do documento WSDL

Minimamente o consumidor do Web Service precisará conhecer a assinatura do serviço (o que entra e o que sai), sua localização e o protocolo a ser usado para enviar a invocação. Um documento WSDL fornece estas informações organizadas em duas seções lógicas : descrição abstrata e descrição concreta [11].

Em Web Services a descrição precisa ser disponibilizada para captura de forma independente de plataforma e linguagem, sendo desta forma que WSDL organiza a informação no documento [11]. A parte de descrição abstrata é composta por quatro elementos XML : elemento types que contem tipos e definições independentes de linguagem e plataforma; elemento messages contém os parâmetros de entrada e de saída para o serviço e descreve as diferentes mensagens trocadas pelo serviço; elemento operations representa a interação particular com o serviço e descreve as mensagens de entrada; saída e exceções possíveis e permitidas durante a interação e elemento portTypes emprega a seção de mensagens para descrever as funções de assinatura (nome da operação, parâmetros de entrada e saída) e representa o conjunto de operações que pode ser executados pelo serviço [11].

Cada Web Service ou ponto de acesso é disponibilizado em um endereço de rede [11]. Um Web Service é capaz de responder a uma solicitação feita através de um protocolo particular ou formato, através do qual uma mensagens e dados são enviados a ele [11]. Estes aspectos de descrição do serviço são específicos da implementação e logicamente agrupados em uma categoria [11]. De modo resumido, a descrição concreta define a especificação de implementação para Web Service [11]. Dois elementos XML são significantes nesta categoria para a descrição do WSDL : elemento bindings especifica a ligação de cada operação descrita na seção do elemento portTypes, associando a descrição abstrata de portType (operações portTypes, mensagens, e tipos de dados) com um protocolo de rede [11]. E o elemento service em adição a informação especifica do protocolo, o documento WSDL deve descrever onde o serviço está disponível ou publicado [11]. A associação entre o elemento binding e o endereço de rede no qual o serviço pode ser encontrado é definido pelo subelemento de service, o elemento port [11] . O elemento service constitui-se em coleção do elemento port que por sua vez descreve um endereço de rede para o elemento bindig [11].

A separação lógica da informação abstrata (como métodos, parâmetros e mensagens de erro) da informação que pode mudar com o tipo de implementação (protocolo de transporte, endereço na rede) permite o reuso das definições abstratas dos serviços através de diferentes implementações do mesmo [11]. Como exemplo podemos citar um mesmo serviço exposto como serviço baseado em HTTP e em SMTP. Em ambos os casos o serviço desempenha as mesmas funções, então a descrição abstrata e os métodos do serviço permanecem os mesmos, podendo ser reutilizados. Somente a descrição concreta, relacionada com informações referentes a especificação da implementação precisa ser trocada e adaptada para atender as duas instâncias nas quais o serviço será exposto [11]. A Figura 3.4.3-1 demonstra, de modo simplificado, representação conceitual do documento WSDL.



NOTA : AS SETAS REPRESENTAM RELACIONAMENTOS POR REFERENCIA ENTRE OS ELEMENTOS E AS CAIXAS RELACIONAMENTO DE CONTAINER.

Fig. 3.4.3-1 representação conceitual do documento WSDL.

Sob a perspectiva das informações que compõe o documento WSDL, oferecemos descrição sucinta do modo como interagem consumidor e fornecedor do serviço, baseados nos sete elementos chaves que compõem a descrição abstrata e a descrição concreta, disponibilizadas no WSDL. Um Web Service expõe, disponibiliza grupos de operações de negócios através do elemento operation para consumidores usarem [11]. Para executar uma operação, o consumidor envia uma mensagem de entrada (input) contendo dados de entrada. Recebendo, em seguida, uma mensagem de saída (output) contendo dados resultantes do processamento da mensagem de entrada ou uma mensagem de falha, no caso de haver ocorrido algum problema de processamento [11].

Mensagens de entrada e de saída podem ter vários itens de dados, denominados na tecnologia WSDL de partes (elemento part) [11]. O protocolo de rede a ser empregado para a troca de mensagens na rede é especificado na descrição concreta pelo elemento binding [11]. O elemento service publica o serviço a ser usado através dos elementos ports que individualmente indicam endereços de

rede onde o Web Service estará localizado [11], aguardando para ser acessado. O elemento port liga-se ao elemento binding através de XML Schema. Um Web Service pode possuir várias portas, sendo que cada porta está ligada a direntes binding de modo que um Web Service pode possuir uma porta que o ofereça em SOAP sobre HTTP e outra com SOAP sobre STMP [11]. A Figura 3.4.3-2 Representa a dinâmica da interação entre serviço e consumidor, baseada nos principais elementos que compõe essa interação.

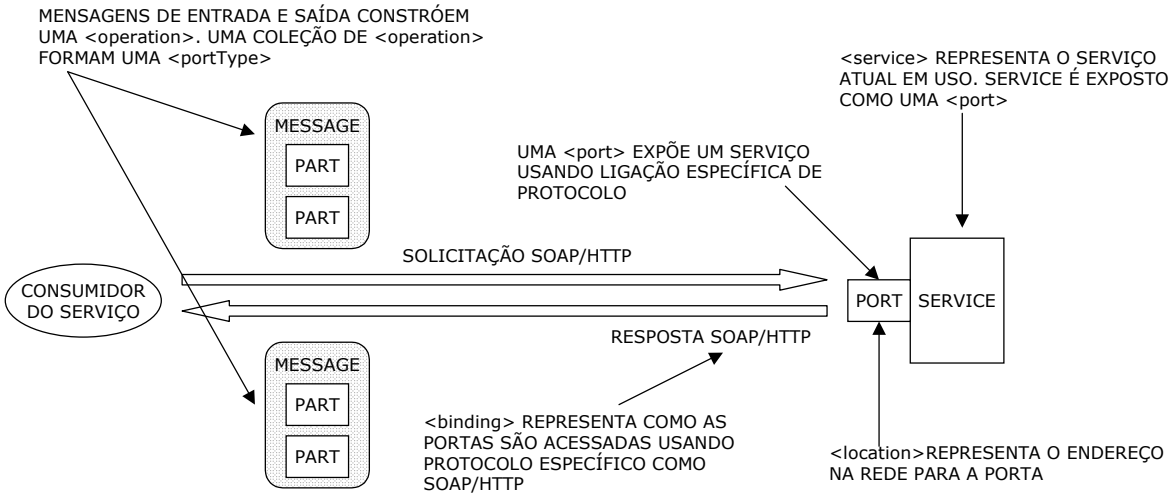


Fig. 3.4.3-2 representação dinâmica da interação entre serviço e consumidor.

A complexidade do documento WSDL está diretamente relacionada com as muitas possibilidades de relacionamentos e ocorrências repetidas que vários dos seus sete elementos principais podem adicionar a um contexto nas descrições abstrata e concreta. Cada elemento binding referencia um elemento portType que por sua vez é comporto do elemento operation e dentro deste elemento operation, existe o elemento message [11]. Cada elemento portType pode conter zero ou mais elementos operation, cada um com elemento message de entrada e de saida. Cada elemento message pode conter zero ou mais elementos parts e cada part constitui-se em algum tipo de dado [11]. O tipo de dado pode ser nativo, da própria especificação do WSDL ou pode ser um tipo de dado definido no elemento type [11]. A Figura 3.4.3.-3 demonstra graficamente o diagrama lógico de classes para os elementos chaves do WSDL.

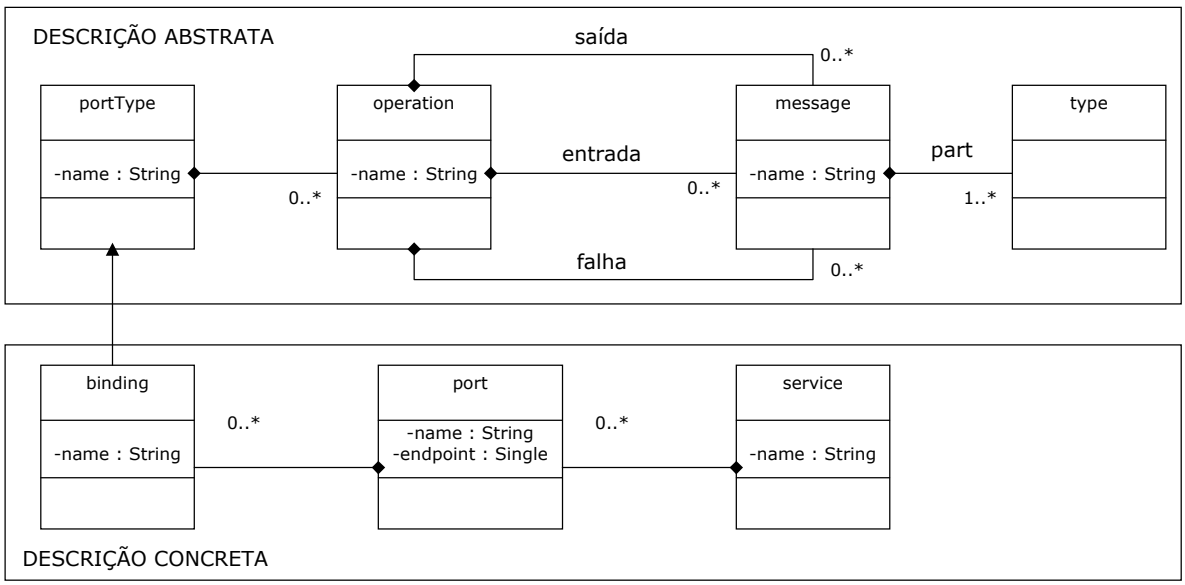


Fig. 3.4.3-3 diagrama lógico de classes para os elementos chaves do WSDL.

3.4.4 – Elementos do documento WSDL

O elemento raiz e principal do documento WSDL é o elemento definitions, a baixo do qual todos os outros elementos são definidos. Este elemento constitui-se o container mais externo do WSDL e contem declarações relevantes de namespace como http://schemas.xmlsoap.org/wsdl/ que descreve quais XML Schemas devem ser usados sobre o documento [11]. A Figura 3.4.5-4 representa o elemento definitions que é composto por um conjunto de seis elementos, pode-se observar que deste grupo, somente o elemento types deve ocorrer uma única vez, sendo que os demais podem ocorrer zero ou n vezes, sendo n um número muito grande.

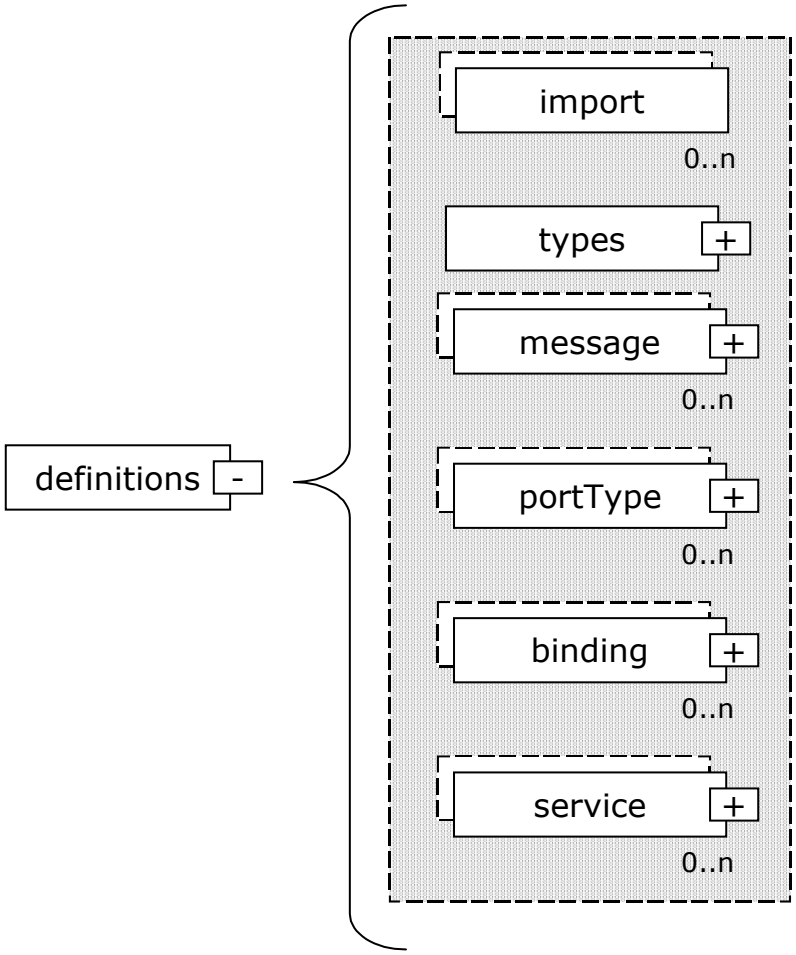


Fig. 3.4.5-4 elemento definitions

Como segundo elemento hierárquico no documento WSDL temos o elemento types que especifica e lista todos os tipos definidos pelo usuário que serão usados pelo Web Services [11]. Além do conjunto de tipos herdados do XML Schemas, empregados com o prefixo xsd, a especificação WSDL reconhece que somente um único sistema de tipos pode não ser capaz de descrever todos os formatos requeridos para mensagens, então disponibiliza o emprego de outras linguagens para a definição de tipos através de mecanismos de extensão [11]. Em termos práticos o elemento types deve ser considerado o local onde os esquemas de tipos são definidos e referenciados [11], entre estes temos o prefixo xsd de uso canônico, herdado das especificações do XML Schemas. Entre as extensões possíveis para tipos a especificação WSDL também designa mecanismos especiais para a manipulação de vetores, similares a descrição usada pela codificação SOAP [11]. Mesmo não especificando valores padrão para tipos vetores, a especificação WSDL introduz o atributo arrayType [11].

A interação entre o consumidor do serviço e o serviço acarreta, naturalmente, a troca de conjunto predefinido de mensagens em dois sentidos. Todas as mensagens que possam vir a ser trocadas entre consumidor do serviço e o serviço, são listadas no documento WSDL empregando-se elementos message. Um elemento message contém a definição abstrata do conteúdo da mensagem [11].

O conteúdo da mensagem depende diretamente da natureza da interação entre consumidor e serviço. Podemos citar algumas da principais modalidades de interação, quais sejam : em uma interação estilo RPC, a mensagem enviada do consumidor para o serviço poderá consistir de zero ou mais parâmetros, então a mensagem poderá ser descrita como uma lista de parâmetros [11].

Em mensagens assíncronas, comumente temos o padrão de pares ordenados compostos de nome e valor [11]. Para mensagens de conteúdo orientado a documento, temos documentos inteiros sendo transmitidos entre consumidor e serviço. Assim, em todas as possibilidades, cada parâmetro da mensagem, denominado de part no WSDL, deve estar associado a um tipo definido previamente no elemento types ou que faça parte do conjunto de tipos herdados da definição XML Schemas.

Cada elemento menssage possui um atributo name que deve conter o nome da mensagem de modo que o mesmo seja único em todo o documento WSDL [11]. Cada elemento message pode conter zero ou mais elementos part [11] que representam os atributos da mensagem, os parâmetros da procedure, no caso de mensagens estilo RPC. Cada elemento part possui o atributo name que deve ser único, o atributo type que deve associar o elemento part a algum tipo XSD (complexo ou simples) ou tipos definidos no elemento types [11].

Como nas linguagens de programação [53] , WSDL permite que mensagens possuam três tipos de parâmetros : entrada, saída e entrada-e-saída. Parâmetro de entrada é aquele recebido do cliente junto com uma mensagem. Um parâmetro do tipo saída constitui-se em uma variável que é inicializada no serviço e enviada para o cliente [11]. Parâmetro do tipo entrada-e-saída é inicializado no cliente, alterado e possivelmente sobrescrito no serviço e enviado de volta para o cliente, em outras palavras , se uma operação recebe um parâmetro que espera ser trocado pelo resultado de uma operação e reenviado para o cliente, este parâmetro é denominado de entrada-e-saída [11]. Em WSDL existe a seguinte regra para a classificação de um parâmetro : se aparece nas mensagens de entrada e saída é parâmetro de entrada-e-saída, se aparece exclusivamente nas mensagens de entrada é parâmetro de entrada porém se aparece somente nas mensagens de saída, então é parâmetro de saída [11]. Em alguns casos, o serviço espera receber os parâmetros em uma certa ordem fixa a qual os consumidores precisam saber para que as mensagens possam ser processadas corretamente [11]. Para atender a este requisito, o elemento operation possui atributo denominado parameterOrder que fornece em uma lista todos os parâmetros na ordem adequada que o serviço espera receber, separados por vírgula, atendendo a assinatura do método que se deseja invocar[11].

O elemento operation define a interação (resultado da troca de uma ou mais mensagens) entre o consumidor do serviço e o serviço de modo abstrato [11], constituindo-se container para todas

as mensagens abstratas que podem ser trocadas em uma interação particular com o serviço [11]. O elemento operation, localizado dentro do portType, contém um elemento input (entrada), um elemento output (saída) e um ou mais elemento fault (falha) que referenciam elementos message [11]. O elemento fault representa o formato abstrato da mensagem para erros ou exceções que possam ocorrer como resultado de operações de saída [11].

Podem haver diferentes tipos de interação entre o consumidor e o serviço, que podem ser descritas pelo elemento operation, entre estas podemos citar : mensagem em um único sentido, solicitação-resposta, resposta-solicitação e notificação. Para o tipo de interação mensagem em sentido único, a operação é somente de envio da mensagem para o serviço e nenhuma resposta é retornado [11] que é usual para o tipo de mensagem "dispara e esquece" no qual o trabalho feito pelo servidor que recebe a mensagem é assíncrono ao do emissor [11]. Em WSDL elemento operation para mensagem em um único sentido não possui o elemento output.

O tradicional caso de interação entre consumidor e servidor baseado no modelo solicitação-resposta, o consumidor envia mensagem ao serviço (solicitação) e aguarda resposta. O serviço pode responder com mensagem de retorno apresentando o resultado da solicitação ou com mensagem de falha, significando que ocorreu alguma coisa errada [11].

No tipo de interação resposta-solicitação, a comunicação ocorre no sentido inverso do tipo solicitação-resposta, neste caso o Web Service envia mensagem para o cliente mensagem de saída e recebe do cliente uma mensagem de entrada. O modo de configurar o elemento operation para este tipo de interação é similar ao anterior, a única diferença é que a mensagem de saída deve ser colocada antes da mensagem de entrada [11], no documento WSDL. Para que este tipo de interação possa tornar-se viável o Web Service precisa conhecer o cliente que vai responder a solicitação e isto pode ocorrer através de inscrição prévia deste cliente como possível respondente da solicitação.

Configurar o elemento operations para o tipo de interação notificação faz com que o Web Service notifique o cliente sobre a ocorrência de alguma espécie de evento no qual o cliente registrou-se previamente como interessado. Do ponto de vista do Web Service, neste tipo de interação não há mensagem de entrada, somente mensagem de saída [11].

O elemento portType combina todas as informações abstratas que juntas descrevem o que o serviço oferece, isto é, a interface abstrata do Web Service [11]. O elemento portType constitui-se o container para as operações (elemento operation) que em última análise é o container das mensagens (elemento message) [11]. De acordo com a especificação, um documento WSDL pode conter um ou

mais elementos portType, porém princípios de boas práticas recomendam fortemente que um documento WSDL descreva apenas uma interface de Web Service[11] .

Cada elemento portType pode ter um ou mais elementos binding, os quais tem por objetivo associar protocolos específicos para a invocação do Web Service [11]. SOAP sobre HTTP pode ser o protocolo mais popular para implementação de Web Services no estilo RPC, mas existem outras possibilidades de protocolos, como SOAP sobre MIME ou HTTP/GET [11]. O importante é que a lista de protocolos que podem ser suportados pela comunidade Web Services não é finita [11]. WSDL dispõe de mecanismos para incorporar informações específicas de protocolo empregando elementos de extensibilidade, os quais produzem gramática de ligação concreta com cada elemento menssage, de cada elemento operation [11]. Atualmente a especificação WSDL define extensões de ligação do elemento binding com pontos de acesso capazes de enteder os protocolos SOAP sobre HTTP, HTTP GET/POST e MIME [11].

Todo elemento binding possui atributo name que indica seu nome único, atributo type liga o elemento binding a respectiva portType e suas operações, descritas nos elementos operation [11]. WSDL define três modos para o elemento binding descrever e expor fisicamente o serviço que será invocado por estes bindings : SOAP bindings que descreve como o serviço pode ser consumido usando mensagens SOAP, MIME bindings que indica como o serviço pode ser consumido enviando mensagens MIME e HTTP bindings que informa como o serviço pode ser consumido com o envio de HTTP GET solicitação e mensagens POST [11].

O SOAP bindings foi definido dentro da especificação WSDL 1.1, o que permite que este binding tenha facilidades e detalhes particulares no documento WSDL [11] como elementos e extensões especialmente criadas para facilitar o uso deste protocolo. Entre as facilidades temos o elmento soap:binding para indicar que o binding que o contém é para o formato de protocolo SOAP. Este elemento deve estar presente se o protocolo SOAP for usado [11].

Para indicar o estilo das operações (elemento operation do elemento binding) a serem disponibilizadas há o atributo style que indica o estilo da operação, podendo variar em orientado a RPC com o valor RPC ou estilo orientado a documento com valor document [11]. O estilo orientado RPC informa que a operação espera como entrada parâmetros e retornará como resposta o resultado do processamento destes parâmetros. Para o estilo orientado a documento, o serviço espera como entrada documentos XML bem formados e válidos e retornará como resposta documentos XML bem formados e válidos [11].

O atributo transport, terceiro atributo do elemento soap:binding, informa o protocolo de transporte, da camada a baixo ao SOAP, que será empregado para transmitir a mensagem e pode ter valores padronizados como http://schemas.xmlsoap.org/http para indicar que o protocolo de transporte é o HTTP ou http://schemas.xmlsoap.org/smtp para indicar SMTP como protocolo de transporte [11]. Outros elementos fundamentais do SOAP binding são soap:operation, soap:body soap:header e soap:fault. O elemento soap:operation constitui-se em uma extensão específica para atribuir valor ao header SOAPAction [11], obrigatório para mensagens SOAP transportadas sobre HTTP.

Para especificar como as mensagens devem ser montadas no elemento Body do SOAP, o elemento binding do documento WSDL, dispõe do elemento soap:body que é usado para mapear as mensagens abstratas de entrada e saída para o protocolo SOAP específico de implementação [11], o protocolo que levará as mensagens pelo fio. Este elemento tem função de definir como as mensagens transmitidas serão codificados (serializados) e decodificados (descerializados) entre o serviço e o consumidor e vice-versa, obedecendo regras de codificação SOAP [11]. O atributo use deste elemento indica que tipo de codificação será empregada, se seu valor for igual a "encoded", então será empregado o esquema de codificação apontado pelo atributo encodingStyle (normalmente este atributo contém http://schemas.xmlsoap.org/soap/encoding, por outro lado, se o valor for "literal" o esquema de codificação a ser adotado é o padrão definido para tipos do XML Schema [11].

Operações em estilo documento não possuem regras de codificação de dados sofisticadas, normalmente os dados são enviados de forma literal e validados no destino via XML Schema [11]. Entretanto, operações que empregam com maior freqüência procedimentos otimizados e sofisticados de codificação são operações orientadas ao estilo RPC [11]. Porém, mesmo não sendo comum, pode ocorrer a combinação entre operações orientadas a documento que empreguem regras de codificação sofisticadas e operações orientadas a RPC que utilizem codificação literal, a mais simples entre as duas [11]. Esta variação pode ocorrer da combinação do atributo style do elemento binding com o atributo use do elemento soap:body. Para a codificação e decodificação das mensagens de falha outro elemento e mecanismo são disponibilizados.

Headers representam o poder de extensibilidade do protocolo SOAP. Qualquer entrada de header que deva ser transmitida como parte do elemento Header, no envelope SOAP, deve ser descrita, especificada e definida pelo elemento soap:header (que faz parte do elemento binding) no documento WSDL [11]. Existe ainda o elemento opcional headerfault que especifica a mensagem de falha correspondente a problemas que podem ocorrer com o processamento da entrada de header [11]. A especificação SOAP exige que mensagens de falhas ocorridas nas entradas de header sejam

retornadas na própria seção do elemento SOAP Header e nunca no elemento Body do envelope SOAP [11].

As mensagens de falhas que podem ocorrer no processamento do conteúdo do elemento Body, são descritas pelo elemento de extensão soap:fault que possui os mesmos atributos do elemento soap:body descrito anteriormente [11]. Cada elemento operations do elemento binding pode ter seus subelementos de entrada, de saída e de falha. No caso do elemento de falha o mesmo pode ser completamente descrito através do elemento soap:fault em aspectos como codificação, decodificação e outros.

Mensagens abstratas descritas no documento WSDL podem ser transportadas pelo protocolo MIME. Especificamente estão definidas três formas, segundo as quais WSDL pode transmitir mensagens ligando-se ao protocolo MIME : multipartes/relacionadas, aplicação/x-www-form-urlencoded, Text/xml [11]. No caso de mensagens que empregam o MIME com a opção de multipartes/relacionadas o elemento mime:multipartRelated funciona como container externo das partes que estão sendo transportadas as quais compõe mensagens (do elemento message) constituídas de texto, imagens, dados binários e outros tipos de informação que tem natureza diferenciada, porém ligação semântica. O elemento mime:multipartRelated fica posicionado sempre dentro dos elementos input e output, para envolver de modo adequado as mensagens a serem transmitidas e recebidas.

Para operações invocadas empregando-se solicitações baseadas em HTTP GET ou POST existem definições WSDL para a ligação de operações (elemento operation do elemento binding). Este aspecto de invocação para Web Services é raramente empregado, porém pode ser útil quando o objetivo é expor partes de Web Services que aceitam quantidade limitada de parâmetros e retornam resultados somente para leitura [11]. Entretanto, com a evolução dos Web Services e das especificações de WSDL, a especificação desta forma de ligação, visa enfaticamente, manter compatibilidade com versões anteriores de Web Services que empregavam este tipo de abordagem. Recomenda-se fortemente que se empregue, para todo e qualquer Web Services, ligações com o protocolo SOAP e não diretamente com HTTP[11].

O local físico do processo onde funciona esquema particular de codificação é indicado pelo elemento port (que está localizado dentro do elemento service em um documento WSDL) [11]. Está é a definição mais concreta, no sentido do concreto para abstrato (de baixo para cima), para este elemento que representa o endereço de rede, o ponto de acesso lógico/físico do Web Service na rede [11]. Para cada elemento binding deve haver um elemento port que informa o nome pelo qual o Web

Service será acessado e conhecido pelo mundo exterior e onde, em que URI, o mesmo poderá ser encontrado.

Na definição WSDL o elemento service comporta-se como container para um ou mais elementos port [11]. A noção de serviço como coleção de portas torna-se bastante útil porque o mesmo Web Service pode ser implementado utilizando-se protocolos diferentes [11]. Nestes casos um elemento portType pode ter vários bindings (onde se define o protocolo) que por sua vez estarão ligados a uma porta cada um, com um elemento service combinando todas as ports no mesmo documento WSDL [11]. Um documento WSDL pode ter mais de um elemento service, porém esta possibilidade contraria princípios de boas práticas [11].

Reusabilidade constitui-se em uma das prioridades da tecnologia WSDL. O elemento import possibilita o referenciamento entre dois documentos WSDL e o estabelecimento do namespace para os elementos importados[11]. Partindo deste princípio partes já testadas e usadas de um documento WSDL como o elemento types ou o elemento service, podem ser reutilizadas em novos documentos WSDL [11]. Porém, possibilidades mais avançadas de reusabilidade podem ser implementadas pelo fato de documentos WSDL representarem concretamente a separação entre a descrição de interface da sua implementação [11]. Com esta separação poderá ocorrer da publicação de documentos WSDL como demanda para implementação, cabendo ao desenvolvedor interessado, construir o Web Service especificado importando WSDL publicado para o qual realizou o desenvolvimento e disponibilizando o Web Service sob medida.

WSDL constitui-se uma tecnologia aberta e extensível, elementos como soap:header, soap:body e soap:fault representam pontos extensíveis do padrão, empregados com funções bem definidas. Existe a possibilidade da criação de elementos extensíveis particulares, próprios de algumas aplicações de Web Services [11]. Este mecanismo de extensão de elementos deve ser empregado para que sejam adicionadas mais informações, métodos complexos e elaborados que permitam descrições de Web Services que possam estender-se para áreas ainda mais amplas e diversificadas da computação [11].

Ao decidir utilizar o expediente dos elementos extensíveis do WSDL o arquiteto do Web Service deve equacionar previamente as seguintes questões : a grande maioria dos clientes potenciais para o consumo do Web Service não conseguiram entender e utilizar os elementos extensíveis particulares e estes terão que ser explicados manualmente e segundo, as ferramentas padrão para WSDL também não reconhecerão estes elementos [11]. Considerando estes dois aspectos, a utilização de elementos extensíveis do WSDL representa flexibilidade e poder para descrição de Web Services com maior complexidade agregada.

3.4.5 – Uso de padrões em WSDL

Existem duas perspectivas para a aplicação de um documento WSDL : o lado cliente do serviço e o lado servidor [11]. No lado cliente, interessado no Web Service descrito, toma o documento WSDL e procura adequar-se as exigências para obter o serviço descrito. Sob a perspectiva do servidor de Web Service, temos que considerar a possibilidade de que o desenvolvedor toma um documento WSDL e a partir deste decide implementar o Web Service descrito. Porém, o mais corriqueiro, no lado servidor, é que o Web Service seja desenvolvido, disponibilizado e durante estas etapas o WSDL seja produzido.

No lado cliente, considerando ambientes onde se emprega RMI ou EJB, antes que o cliente possa invocar os métodos expostos por um Web Service no estilo RPC, faz-se necessário ter conhecimento de informações como : localização (endereço de rede) do serviço, protocolo que o serviço utiliza (para que a solicitação possa ser decodificada corretamente) e os métodos com suas respectivas assinaturas [11]. Todas estas informações podem ser encontradas no documento WSDL que descreve o serviço. A obtenção do documento físico WSDL pode ser concretizada através de referência fornecida pelo diretório que anuncia e publica apontando para uma bem formada URL que hospeda o arquivo físico do documento WSDL ou ainda através de algum outro mecanismo previamente divulgado ou acordado entre fornecedor e possíveis consumidores, como por exemplo correio eletrônico [11].

Segundo a perspectiva dos produtores do serviço, implementadores do Web Service, responsáveis em disponibiliza-lo na rede, podem adotar uma das seguintes estratégias : a partir de uma aplicação já em funcionamento, expor parte desta como Web Service implementando a camada de interface SOAP [11]. Como Segunda estratégia importar um documento WSDL e providenciar a implementação para as exigências descritas e publicar o serviço [11].

Tanto no lado do cliente como no lado do desenvolvedor existem ferramentas capazes de ler documentos WSDL e produzir as respectivas implementações em diversas linguagens [11]. Entretanto temos que considerar os seguintes aspectos relevantes ao decidirmos empregar estas ferramentas para utilização com a linguagem Java : mapeamento de parâmetros para tipos de dados, de Java para XML Schema e XML Schema para Java, mapeamento do elemento portTypes para interfaces e classes, geração de ligações baseadas nos conteúdos dos atributos style e use (considerando as possíveis combinações RPC/encoded, RPC/literal, document/encoded e document/literal) e a

importação da informação de ponto de acesso, endereço de rede, para dentro do código fonte gerado para as classes proxy no cliente [11].

3.4.6 – Ligação antecipada ou tardia

O conceito de ligação antecipada ou tardia esta relacionado diretamente com o modo pelo qual consumidores de Web Services passam a tomar conhecimento dos detalhes necessários para a utilização do serviço com o qual pretendem interagir. Considerando que o documento WSDL contém as informações necessárias para esta interação, consumidores podem empregar alguns padrões de arquitetura baseados neste conceito, para a ligação com o serviço, na ocasião de invocação dos mesmos [11].

Para Web Services com interfaces e localização não voláteis, o conceito de ligação antecipada apresenta-se como a solução canonicamente mais indicada [11]. Genericamente, este conceito funciona a partir da importação do documento WSDL para dentro do ambiente de desenvolvimento, no qual são gerados, através de ferramentas automatizadas, artefatos (stubs, mapeamento para tipos de dados, código fonte e proxy cliente) adequados para que o Web Service seja acessado pela aplicação, eliminando qualquer necessidade de inspeção ou leitura do documento WSDL para os acessos futuros ao Web Service por ele descrito [11]. Este procedimento é rotineiramente empregado para o consumo de Web Services baseados no estilo RPC [11].

A descrição genérica da ligação antecipada possui três variações que estão relacionadas com o quando (em relação a fase de implementação da ligação antecipada) as partes das descrições abstrata e concreta do documento WSDL são usadas pelo consumidor para efetivar a ligação antecipada [11]. É importante lembrar que o elemento portType (descrição abstrata) define a assinatura funcional do Web Service [11]. As variações são : ligação estática em tempo de compilação, ligação estática em tempo de publicação (deploy) e ligação estática em tempo de execução. Na variação da ligação estática em tempo de compilação ambas descrições abstrata e concreta são completamente conhecidas e empregadas diretamente [11] acreditando-se que não sofrerão alterações e mudanças por algum tempo. Neste caso elementos como port, portType e o local do serviço são conhecidos e informados em tempo de compilação [11]. Tipicamente o consumidor do serviço desenvolve um stubs [11].

Na segunda variação de ligação antecipada temos a ligação estática em tempo de publicação (deploy), na qual o elemento portType que descreve a assinatura funcional do serviço é conhecido durante o tempo de desenvolvimento, porém a localização, o endereço na rede do serviço

somente será conhecido no tempo de publicação (deploy). Isto significa que na publicação (deploy) do Web Service o consumidor será obrigado a procurar no WSDL disponibilizado o elemento port e obter a localização, o endereço do serviço na rede [11]. Tipicamente o consumidor é desenvolvido empregando-se a geração de stub no tempo de compilação, para esta modalidade de ligação antecipada [11] que procura o endereço do Web Services em tempo de execução, consultando o WSDL.

Temos a terceira variação de ligação antecipada que chamamos de ligação estática em tempo de execução. Neste caso específico a única parte conhecida em tempo de desenvolvimento e, por assim dizer, fixa é a descrição abstrata, o elemento portType que informa a assinatura funcional do serviço. Informações como o elemento port e o endereço, o ponto de acesso, o local do Web Service, só serão conhecidos quando o consumidor tiver a necessidade de utilizar o Web Service [11]. O cliente para este tipo de ligação estática também será baseada em stub. Este estilo de ligação antecipada aplica-se a segmentos onde existe padronização na oferta de serviços, que possuem a parte de descrição abstrata padronizada, ficando a parte de descrição concreta para que o consumidor possa realizar a escolha segundo a disponibilidade e outros critérios na ocasião de uso do Web Services[11], está realidade ainda não é concreta, porém a flexibilidade oferecida pela tecnologia WSDL permitirá sua implementação.

A outra maneira de invocar Web Services é baixar e examinar o documento WSDL em tempo de execução (em pleno vôo) e dinamicamente invocar e utilizar o serviço selecionado [11]. A esta modalidade denominamos de ligação tardia. A adoção desta modalidade é indicada como a forma mais robusta para a utilização de Web Services que sofrem alterações e trocas imprevisíveis ou possuem características variáveis [11]. Este método pode, de modo geral, apresentar baixo desempenho em relação aos métodos de ligação antecipada [11]. Como vantagem mencionamos que trocas e mudanças nas assinaturas funcionais, de endereço na rede e outras produziram baixo impacto sobre a aplicação consumidora do Web Service, isto ocorre devido ao baixo acoplamento entre as partes, consumidor e fornecedor de Web Service [11].

Dois métodos podem ser empregados para acessar dinamicamente Web Services no estilo ligação tardia : invocação dinâmica de interface (DII Dynamic Invocation Interface) e proxies dinâmicos. Em ambos os casos o WSDL é usado intensamente, com a vantagem de que o cliente necessita saber muito pouca informação sobre o Web Service em tempo de compilação e algumas delas são extraídas em tempo de execução simplesmente apontando o cliente para a URL onde o WSDL encontra-se hospedado [11].

Temos ainda a possibilidade de implementar ligação tardia utilizando os métodos invocação dinâmica de interface (DII Dynamic Invocation Interface) e proxies dinâmicos a partir de duas estratégias básicas : ligação dinâmica e ligação dinâmica com conhecimento de ponto de acesso. No caso da ligação dinâmica, nenhuma informação sobre o Web Service é conhecida, não se tem informação sobre a assinatura funcional (elemento portType) e nem conhecimento de onde o Web Service se encontra (endereço na rede) no tempo de compilação. O consumidor é responsável por localizar o serviço desejado, realizar a introspeção do elemento portType (do documento WSDL) e subseqüentemente invocar o serviço. Ambos os métodos invocação dinâmica de interface (DII Dynamic Invocation Interface) e proxies dinâmicos podem sem empregados neste caso [11].

A estratégia para ligação tardia denominada ligação dinâmica com conhecimento de ponto de acesso constitui-se variação menos radical da estratégia anterior, considerando que neste caso o localização, o ponto de acesso, o endereço da rede onde se encontra o WSDL é conhecido em tempo de desenvolvimento, porém a descrição funcional (o elemento portType) não é [11]. Neste caso o consumidor tenta acessar o endereço de rede para obter o portType ou envia para o endereço de rede algum modelo prévio de portType para verificar se o mesmo é suportado [11].

Diante das questões e dificuldades superficiais expostas, a ligação tardia a Web Services, bem como a invocação dinâmica, são consideradas padrões de arquitetura pouco práticas e de difícil implementação para uso industrial e cotidiano, por enquanto[11].

3.5 – UDDI

Como solução para muitos problemas encontrado na consecução de negócios empregando a Internet em B2B (Business to Business), a indústria, liderada pelas empresas Ariba, IBM e Microsoft, definiu UDDI (Universal Description, Dicovery Interface) que constitui-se a primeira iniciativa implementada e mantida pela própria industria através de corporações, vendedores de softwares e consórcios de segmentos da indústria e comécio [11]. UDDI foi criada para oferecer soluções rápidas para muitas das questões e empecilhos que limitam a rápida adoção do comércio via Internet [11]. Ao contrário de SOAP e WSDL que são capitaneadas e mantidas pelo W3C, a especificação UDDI é independente e mantida por um grupo de empresas do mundo dos negócios [11].

Constituindo-se em um conjunto de especificações, UDDI define o que deve ser feito e como deve ser feito, independente de plataforma, para que negócios possam descrever publicamente seus serviços disponíveis, localizar outros serviços do seu interesse e compartilhar informações sobre pontos convergentes em um registro global [11]. Do ponto de vista operacional, UDDI pode ser entendido como mecanismo que possibilita as organizações localizar outras organizações e conduzir transações de negócio de forma muito rápida, empregando padrões estabelecidos e aceitos por todos os participantes [11]. Por analogia, poderia ser um grande mercado virtual de possibilidades para negócios.

No segmento da indústria que deu origem ao UDDI, o objetivo constitui-se em acomodar nesta especificação as seguintes características que devem atender as necessidades do B2B tais como : descobrir o padrão mais adequando para a condução de negócios na Internet, entre os milhões de padrões que existem hoje; criar um padrão aceito firmemente pela indústria, com uma abordagem padronizada que possa realmente chegar a atender a parceiros e clientes em termos de informação sobre serviços fornecidos e consumidos e que possa transformar-se em método preferencial para a integração entre sistemas e aplicações diferentes; caracterizar como transações de negócios são usadas no comércio, de modo que um parceiro preferencial possa ser selecionado através de meios eletrônicos [11].

Muitos problemas do B2B encontram solução no UDDI. Pela perspectiva da aplicabilidade final para corporações usuárias desta especificação, com UDDI é possível a um negócio encontrar um potencial parceiro, independente da escolha de padrões tecnológicos e/ou protocolos [11].

3.5.1 – UDDI empregado para encontrar Web Services

Um registro UDDI contém informações sobre negócios, dados da organização e dos serviços que ela oferece. De modo estrutural, o diretório de UDDI contém referências, para segmentos específicos da industria indicando serviços suportados e requeridos por estes segmentos, taxionomias e sistemas de identificação[11]. Consumidores potenciais do serviço podem tentar localizar registros de negócios, empresas, registros de serviços ou registros de tipos de serviços [11]. Para classificar, categorizar seus registros, UDDI emprega os padrões de taxionomia da indústria e outros esquemas de classificação como D-U-N-S, código SIC e outros utilizados para classificar empresas, segmentos de atividades e serviços oferecidos como seu acervo de busca[11]. Todas as APIs, bem como a definição de UDDI, são especificados em XML, empacotados dentro de envelopes SOAP e utilizam o HTTP como protocolo de transporte [11].

3.5.2 – Registro UDDI

Um registro UDDI assemelha-se muito a um motor de pesquisa da Internet, como Google, AltaVista, Miner e outros. Entretanto a literatura faz questão de compará-lo a uma lista telefônica, o qual disponibiliza a possibilidade aos potenciais consumidores de serviço de localizar facilmente as entidades de negócios desejadas e os serviços fornecidos por estas [11]. O registro UDDI é constituído por três grandes partes ou divisões : As páginas brancas, as páginas amarelas e as páginas verdes. Nas páginas brancas são encontradas informações que permitem ao consumidor verificar endereço, informação para contato com a empresa, tomar conhecimento dos identificadores utilizados pelo negócio [11]. Consultar estas páginas é similar a consulta que se realiza a lista telefônica quando se tem apenas o nome da empresa e se deseja obter o endereço e o telefone da mesma[11].

A segunda parte do registro UDDI é constituída pelas páginas amarelas que incluem informação de classificação baseada na taxionomia padrão da industria ou segmento que a empresa e os serviços, por ela, oferecidos fazem parte [11]. A implementação típica de páginas amarelas emprega geralmente a estrutura de par ordenado nome-valor [11]. A metáfora empregada no mecanismo de páginas amarelas permite que qualquer termo válido da taxionomia seja associado a qualquer página branca de negócio [11]. As páginas amarelas também possuem dispositivos que consideram uma espécie de taxionomia geográfica[11] afim de possibilitar o georeferenciamento como extensão dos serviços disponibilizados pelo registro de UDDI.

Os serviços na Internet que são fornecidos pelas empresas das páginas brancas, são descritos nas páginas verdes, terceiro e último componente do registro UDDI. Nesta página encontra-se a lista completa dos serviços oferecidos descritos por extenso acompanhados de referências para processos de negócios que os realizam.

Quatro tipos de informações, entidades lógicas, compõem o registro UDDI : entidade de negócio, serviço de negócio, especificação de ponteiros e tipos de serviços. Todas as entidades de negócio possuem identificador único, nome do negócio, informações simples de contato, descrição resumida sobre o negócio, lista de categorias que descrevem e classificam o negócio e uma URL que aponta para informações adicionais sobre o negócio [11].

A segunda informação que compõe o registro UDDI é o serviço do negócio que inclui a descrição do serviço disponibilizado, lista de categorias que descreve e classifica o serviço e uma URL com informações adicionais sobre o serviço[11].

Cada entidade de serviço de negócio inclui a lista de ligações para templates (modelos) que apontam para informações adicionais sobre o serviço. A ligação com o template pode apontar para a URL que disponibiliza informações de como o serviço pode ser invocado. A entidade especificação de ponteiros, terceiro componente do registro UDDI, contém informações que associam o serviço com um tipo de serviço, outra entidade do registro UDDI [11].

A padronização, a tipificação de serviços pode ser feita pela entidade tipos de serviço (Service Type, Technical Model, abreviado por tModel) que compõe o registro do UDDI [11]. O tModel define o tipo de serviço. Considerando que várias empresas oferecem serviços dentro dos tipos definidos pelo tModel. Este componente define a informação que cada serviço oferecido dentro do padrão deve conter o nome tModel, o nome da organização utilizado pelo tModel, as categorias que definem o serviço e os ponteiros para as especificações de tipo dos serviços incluindo as definições de interface, protocolos de mensagens e formatos e protocolos de segurança. Tipicamente o tModel refere-se a um documento WSDL para um serviço [11].

Invocando o paradigma da arquitetura orientada a serviço que emprega a bordagem de procurar, ligar e executar o serviço requerido [11]. Aprendemos que em nenhuma abordagem desta família de arquitetura faz-se necessário a presença da função catálogo de serviços. Assim, a tecnologia UDDI refere-se e encarrega-se exclusivamente do aspecto procura, dentro desta arquitetura[11]. Entretanto, o emprego de UDDI não está limitado somente a Web Services, por projeto, o mesmo pode hospedar entidades relacionadas a outros tipos e espécies de serviços [11].

3.5.3 – UDDI na tecnologia de Web Services

Podemos descrever possíveis interações da tecnologia Web Services com UDDI sob quatro aspectos diferentes de atores que podem fazer uso da estrutura disponibilizada : primeiramente, fornecedores de serviços em geral e companhias de software definem especificações direcionadas a segmentos verticais da industria e do comércio e os registram no UDDI. Este passo pode ser denominado como registro de tModels [11].

Em segundo, organizações registram descrições de seus negócios e atividades fins e dos serviços que as mesmas oferecem. Cada inclusão é assinalada com um UUID (Unique Universal Identifier). A garantia do UUID é que o mesmo será imutável e único no transcorrer do tempo e do espaço [11]. O formado do UUID é estranho a compreensão humana, pois o mesmo apresenta-se como string, no formato hexadecimal em algo do tipo C1B9EF7A-322F-332B-8B3C-3214DA2E5EE1, o UUID é bastante significativo e importante no contexto de registro em que está inserido [11]. Porém, negócios e serviços registrados em UDDI diferentes, receberam UUID diferentes que dependem de seus próprios mecanismos de UDDI [11].

O terceiro aspecto de interação com o UDDI é o das aplicações, motores de pesquisas e intermediários que utilizam os registros disponibilizados pelos UDDI para procurar, buscar serviços que satisfaçam e atendam a critérios de busca [11]. Os UDDIs podem ser pesquisados, consultados e perguntados empregando-se qualquer esquema de classificação e diferenciação relevantes [11].

Após o serviço ser encontrado, com o auxílio do UDDI, o mesmo poderá ser ligado (que refere-se ao ato de haver uma preparação para prévia para a utilização do serviço) e então executado por invocação da interface apropriada [11]. Desta forma, temos a interação simplificada do UDDI com a tecnologia Web Services.

3.5.4 – Provedor de UDDI

Desempenhando papel fundamental para infra-estrutura de Web Services, UDDI disponibiliza mecanismo para organização e gerenciamento de serviços [11] em domínios de atividades e conhecimentos específicos na Internet. As funções básicas do provedor de UDDI são anunciar, divulgar e fazer publicidade dos serviços publicados nele, possibilitar a procura e a descoberta de serviços para clientes potenciais e armazenar informações sobre empresas e serviços prestados na Internet [11].

O provedor UDDI disponibiliza um modo consistente para que organizações e empresas anunciem e divulguem serviços, fazendo com que estes se tornem disponíveis para outras empresas e consumidores em potencial de forma geral[11]. Empresas e consumidores necessitam de capacidade e conhecimento para descobrir serviços contidos no provedor de UDDI. As informações neste provedores estão persistidas nas camadas de armazenamento em XML e outros mecanismos [11].

Funcionando como encanamento entre fornecedores de serviços e potenciais consumidores, os provedores de UDDI disponibilizam infra-estrutura necessária para que ocorra o encontro entre as partes interessadas, que não necessitam recriar nenhum mecanismo particular para permitir a interoperabilidade entre elas [11]. A infra-estrutura do provedor UDDI foi projetada para suportar o armazenamento de metadados, esquemas, interfaces de programação, autenticação baseada em perfil e conectividade com banco de dados [11]. Neste cenário um serviço pode ser localizado tanto através de programação, onde um programa se comunica com o UDDI e realiza a pesquisa desejada, ou através de interação humana, via páginas Web [11].

Comportando-se como registro, UDDI permite a inclusão de serviços, busca e descoberta de metadados e classificação de entidades em categorias predefinidas [11]. Entretanto, o provedor de UDDI não pode ser considerado repositório segundo o ponto de vista de que não há a possibilidade de armazenamento de recursos XML como WSDL ou definições de processos de negócios que são necessários para a formação de acordos como resultados de negociações, pois não existem referências a estas funções no modelo de segurança [11]. O comportamento de repositório deve incluir portanto armazenamento de dados, incluindo documentos XML, armazenamento esquemas e outros tipos de dados e disponibilizar extensivos mecanismos de classificação e ordenação [11].

O termo registro, quando empregado em conjunto com a tecnologia Web Services, quer significar recursos compartilhados entre empresas e parceiros que podem ser acessados como Web Services por meio serviços dinâmicos de baixo acoplamento [11]. Podemos considerar o registro como um catálogo de itens e o repositório como o local onde os itens são armazenados [11].

Para atender ao caráter dinâmico da oferta de Web Services, o provedor UDDI disponibiliza mecanismos de manutenção dos registros que hospeda através de páginas de administração[11] ou de APIs que podem ser acessadas por meio de programação. Cada provedor UDDI determina como irá funcionar sua estrutura de segurança, autenticação e registro [11]. Em termos de segurança o provedor UDDI pode determinar suporte a autenticação empregando opções como LDAP, Active Directory ou qualquer outro esquema particular de credenciamento [11].

Aspecto fundamental dos provedores de UDDI é a publicação das interfaces de serviço, documento padronizado que permite a invocação de um determinado serviço [11]. Uma interface de serviço pode ser descrita em várias linguagens, mas na tecnologia Web Service, emprega-se XML e WSDL [11]. Podemos descrever uma interface utilizando vários métodos entre eles a prosa ou uma linguagem formal [11]. Na arquitetura orientada a serviço, o método recomendado para descrição de

interface de serviço é XML. Uma descrição de interface de serviço pode referenciar outra descrição de interface de serviço através do elemento import [11]. Um documento de descrição de interface de serviço contem o funcionamento implementado do serviço [11] que estará disponível para ser utilizado, desde que o cliente potencial comunique-se com o serviço seguindo a descrição da interface deste serviço.

3.5.5 – Modelo de informação estruturada UDDI

Em seu núcleo primário UDDI é composto por quatro tipos de estruturas : entidade de negócio, entidade serviço negócio, template de ligação e tModel [11]. Existe a possibilidade para a definição de estruturas adicionais cada uma das quais é usada para representar tipos de dados específicos e organizadas em relacionamentos bem definidos.

A cardinalidade dos relacionamentos entre as estruturas do núcleo de um UDDI, pode ser resumida da seguinte forma : cada entidade negócio pode conter uma ou mais entidades serviço que pode conter uma ou mais instâncias de template de ligação [11]. Cada estrutura do core é identificada de forma única baseada no uddikeys (conteúdo da identificação produzido pelo UDDI) [11]. Por padrão, todos os elementos do núcleo são incluídos prontos para publicação, porém existe a possibilidade extensão, dos usuários poderem oferecer suas próprias estruturas como desafio [11]. Cada uma das estruturas possui uma atributo que é do tipo key: entidade negócio possui o atributo businessKey, entidade serviço negócio possui o atributo servicekey e assim por diante com as demais entidades componentes do núcleo [11]. A Figura 3.5.5-1 representa o modelo de informação estruturada usado no UDDI.

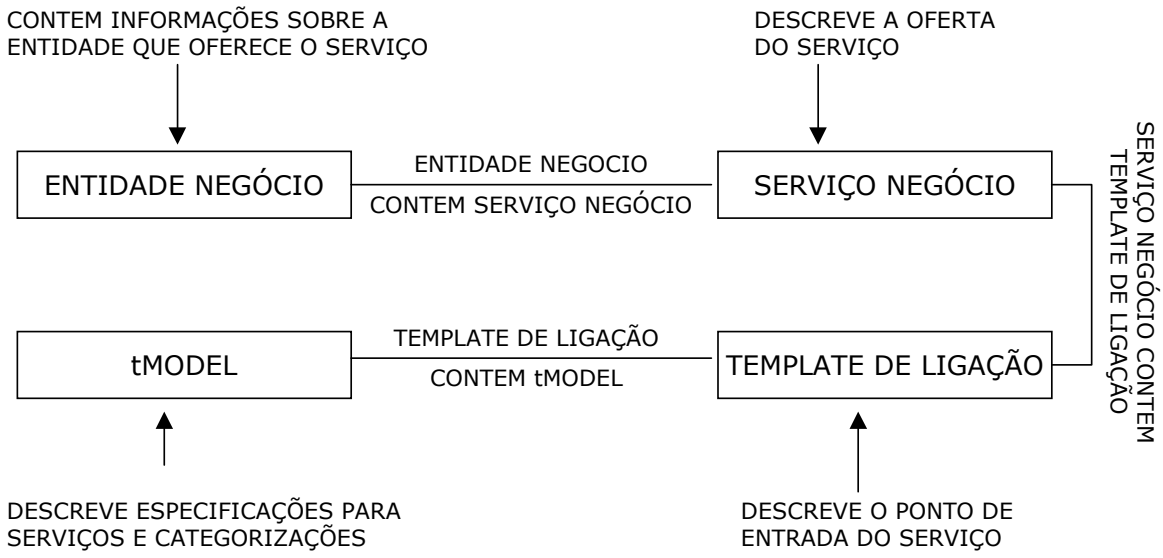


Fig. 3.5.5-1 modelo de informação estruturada usado no UDDI.

A entidade negócio constitui-se em estrutura que contém informações particulares sobre a organização do negócio e também mantém referências sobre os serviços que o negócio oferece. A entidade negócio ocupa o topo, o mais alto lugar na hierarquia e contém informação descritiva [11]. Esta entidade é identificada de forma única no UDDI pelo atributo businesskey, caso este atributo não seja informado, o mesmo é automaticamente gerado pelo UDDI na ocasião da publicação do registro [11]. A Figura 3.5.5-2 representa esquema com a entidade negócio e todos os seus filhos possíveis.

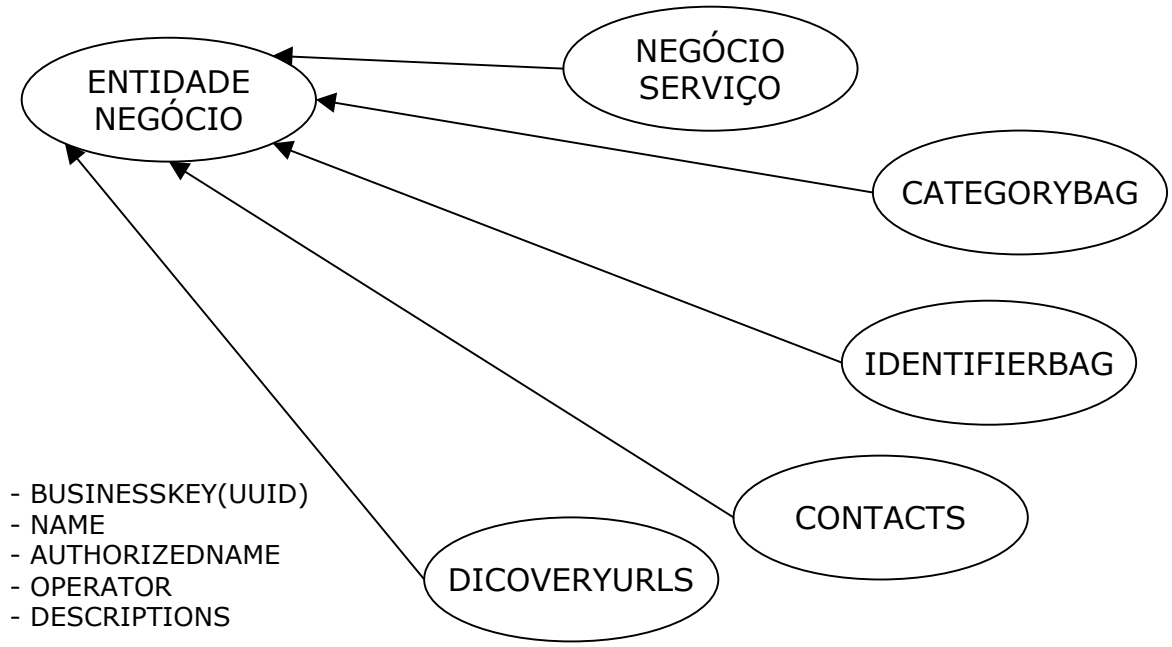


Fig. 3.5.5-2 entidade negócio

Na estrutura do núcleo UDDI, a entidade serviço negócio indica o serviço lógico e mantém a informação descritiva sobre o Web Service disponibilizado em termos de negócio,

informando em linguagem fluente e por extenso como o Web Service funciona e qual o objetivo final prático do mesmo [11]. Considerando a estrutura de dados do UDDI, a entidade serviço negócio é filho da entidade de negócio que disponibiliza o Web Service. Informações adicionais de como uma entidade serviço negócio pode ser instanciada estão contidas no template de ligação [11]. Cada entidade serviço negócio possui um identificador único atribuído pelo próprio UDDI que o usuário que publica o registro não pode alterar. Além deste identificados a entidade serviço negócio possui a chave de identificação do seu pai, a entidade negócio, que também não pode ser alterada [11]. Para representar a entidade serviço negócio e seus possíveis filhos oferecemos a Figura 3.5.5-3 a seguir.

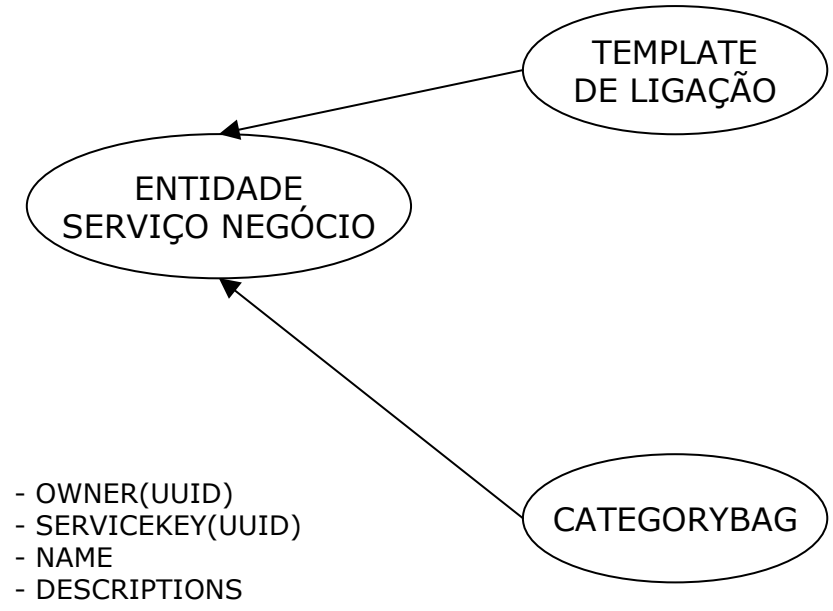


Fig. 3.5.5-3 entidade serviço negócio

A informação necessária para que o cliente invoque o Web Service previamente encontrado no UDDI, está disponível na estrutura template de ligação, que especifica a informação para um Web Service particular e indica a URL onde o serviço pode ser acessado [11]. Pode conter referencias sobre tModel, bem como especificações mais detalhadas sobre configurações do serviço. A entidade template de ligação é filha (hierárquica) da entidade serviço de negócio [11]. A função chave do template de ligação é oferecer ao serviço a possibilidade de expor quais transportes o serviço transporta ou seja um serviço pode suportar vários protocolos de transporte, entre eles HTTP, HTTPS, SMTP e outros, e a entidade template pode informar quais, se for o caso[11]. A entidade template de ligação e seus possíveis filhos é demonstrada esquematicamente na Figura 3.5.5-4.

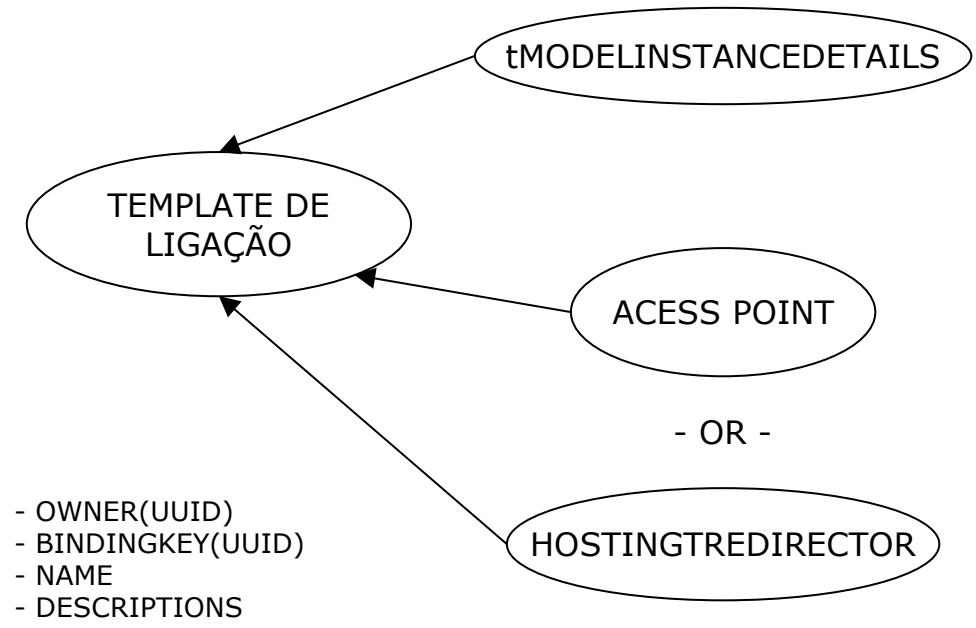


Fig. 3.5.5-4 entidade template de ligação

Uma das necessidades das organizações fornecedores de Web Services é a de possuirem mecanismo capaz de publicar informações sobre especificações e controlar, de modo seguro e prático, as respectivas versões usadas pelos seus serviços [11]. WSDL constitui-se uma das especificações de serviço que necessita empregar esta abordagem [11]. Esta necessidade ocorre porque é normalmente possível que hajam muitas revisões de especificações ativas ao mesmo tempo, aumentando-se a necessidade de distinguir, publicamente as diferentes especificações de tal modo que as especificações ainda em uso possam ser descobertas e usadas por novos possíveis clientes [11]. As informação sobre as especificações contém metadados e são chamadas tModel que é a estrutura no modelo UDDI projetada para suportar esta característica [11].

A estrutura tModel disponibiliza abordagem que permite o reuso e a padronização do modo de gerenciar versões de especificações com Web Services. O conceito que sustenta tModel é útil para a localização e descoberta de informações sobre serviços expostos para amplo uso [11]. A padronização de descrição para Web Services importada por tModel permite que os mesmos sejam agrupados, semanticamente, de modo que a busca por similaridades retorna serviços com especificações similares e muito próximas.

No caso de Web Services, tModel armazena referências aos WSDL no próprio UDDI. Um tModel contém metadados sobre documentos técnicos e uma chave que identifica unicamente o tModel. UDDI utiliza o tModels como modelo de informação para armazenar seus próprios metadados sobre seu modelo de informação que gerência. Podemos citar os seguintes usos de tModels que armazenam metadados do UDDI : descrição de transportes como HTTP, HTTPS e SMTP em

estruturas como (uddi-org:types), formatos diversos de endereços postais (tModel suporta um formato para cada país, bem como as diferentes línguas dos países) e categorizações, identificadores e namespaces [11]. O emprego do tModel permite a utilização do UDDI para representar dados e metadados sobre serviços, entidades negócio e outras partes mais [11]. Adaptada de [12] apresentamos esquema simplificado da estrutura interna de um tModel e seus principais filhos na Figura 3.5.5-5.

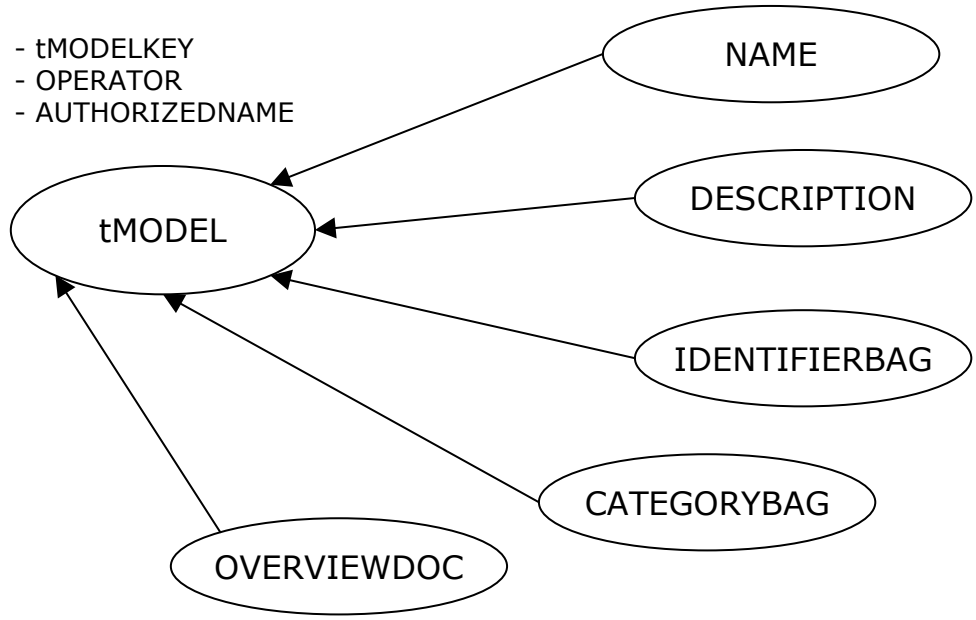


Fig. 3.5.5-5 tModel e seus principais filhos.

3.5.6 – Categorização de Web Services

Uma dos principais objetivos de se empregar um registro, é a manutenção de informações sobre serviços de modo a permitir aos clientes em potencial fácil pesquisa e localização desta [11]. Apenas a manutenção do registro, sem a adequada classificação taxionômica dos dados, não produz resultados ideais [11]. Caso um cliente não consiga localizar rápida e eficientemente um serviço desejado, o uso de um UDDI pode tornar-se um transtorno [11].

Para tornar o emprego e a utilização do UDDI mais adequado, pode-se utilizar taxionomias para categorizar, classificar serviços e outra entidades registradas [11]. Entidades podem ser classificadas com base em sistemas canônicos da administração e formalização comercial e industrial, em seus vários segmentos e cadeias produtivas, surgidos antes mesmo da Internet. Entre estes sistemas temos, para a realidade internacional, classificações como o D-U-N-S, UN/SPC, código SIC e outros [11], no caso de implementação de um UDDI brasileiro poderíamos empregar a mesma classificação utilizada pelo CNPJ. Esquemas de classificação adicionais foram criados para especificar

informações geográficas ou o agrupamento entre membros participantes de uma mesma organização. A classificação de entidades evolui constantemente adaptando-se as novas necessidades dos negócios.

O sistema de classificação do UDDI deve ser empregado em todas as quatro entidades que fazem parte do registro. Uma recomendação genérica das normas do UDDI é a de que todos os serviços sejam rigorosamente classificados antes de serem publicados para utilização [11]. Não importa que o novo serviço que esteja sendo publicado tenha classificação baseada nos padrões correntes ou que seja classificado em um sistema particular, torna-se vital para toda e qualquer informação no UDDI possuir uma classificação específica e adequada [11].

Disponibilizando APIs de busca, UDDI permite diferentes esquemas de classificação. Uma pesquisa pode conter uma coleção de critérios utilizando classificações múltiplas como códigos industriais, códigos de localização geográfica e outros critérios [11]. Com este tipo de facilidade podemos ter pontos de entrada com valor agregado para motores de pesquisa como Google e AltaVista [11]. UDDI pode ainda empregar critérios de classificação para checar a validade de algumas buscas, como por exemplo, verificando se o estado informado, de um determinado país, faz parte da lista de classificação de estados daquele país [11]. Como base da especificação de UDDI três classificações canônicas, de alcance internacional, devem ser suportadas por padrão : ISO 3166 : geográfico, NAICS : industrial e UN/SPC : produtos e serviços [11].

Cada UDDI pode prover seus próprios esquemas adicionais para categorias de classificação. Esta opção esta relacionada com os tipos de negócios, serviços a serem oferecidos e mercados a serem atingidos [11]. O esquema de classificação pode estender-se para os elementos internos do registro e oferecer a possibilidade de categorizações particulares como por exemplo por preço, disponibilidade, qualidade de serviço e outros fatores [11]. Algumas organizações criam seus próprios critérios de categorização baseados em lista de transportadores preferenciais, lista de fornecedores mais confiáveis ou lista de clientes com melhor índice de crédito [11].

Mecanismos para garantir a qualidade e a integridade das categorizações são implementados no UDDI. Entre estes mecanismos estão a garantia de que registros a serem categorizados receberão somente valores válidos, previamente inseridos nas categorizações possíveis [11] . Para tornar o tempo de resposta menor e manter o UDDI com o máximo de agilidade, antes de iniciar uma busca todos os valores de categorias informados nesta busca são verificados quanto a pertencerem ou não a pelo menos uma categoria válida [11].

A decisão de criar uma categoria própria pode ser uma necessidade especifica de determinada organização que possui flexibilidade no UDDI. Um nova categoria pode relacionar-se com categorias já existentes, no sentido de agregar a categoria já existente a nova [11]. Sendo assim é perfeitamente possível definir-se uma nova categoria e adicionar a mesma a restrição de que ela aplica-se somente a alguns países da América Latina. Outra flexibilidade para as novas categorias personalizada é a derivação a partir de categorias existentes que podem ser estendidas [11] para conter mais atributos e mais informações, tornando-as com um grau de precisão maior que a categoria original.

3.5.7 – Identificadores UDDI

Entidades registradas no UDDI tem a possibilidade de ser diferenciadas através de informações provenientes de identificadores. O principal objetivo de empregar-se vários identificadores em registros UDDI é permitir a identificação única do negócio, baseado em esquemas de categorias definidas [11]. Normalmente os vários identificadores de negócio estão anexados na entidade negócio ou em entidades tModel [11]. Entre os identificadores formais que podem ser atribuídos a empresas para que possam ser registradas e localizadas temos o número D-U-N-S, números do governo para a cobrança de taxas e impostos ou a sua identificação pública na bolsa de ações [11].

Em casos de UDDI privados pode-se criar vários identificadores internos deste UDDI, porém, em ocasiões onde ocorra a necessidade de troca de informações com UDDI externos, somente os identificadores comuns e canônicos poderão ser usados e reconhecidos [11]. O número de identificadores que um registro UDDI pode ter é teoricamente ilimitado, estando vinculado somente as especificações internas determinadas pela administração deste [11]. Cada identificador deve estar concretamente ligado a uma categoria válida no UDDI.

3.5.8 – Relacionamento entre entidades de negócio

Grandes conglomerados de empresas tem a necessidade de registrar várias filiais como entidades de negócio individuais e manter uma ligação entre elas [11], de modo a refletir a realidade concreta de que fazem parte do mesmo grupo empresarial ou holding. Em resposta a esta necessidade das grandes corporações, verdadeiros usuários dos UDDIs, existe a API de busca e pesquisa denominada find_relatedBusinesses que retorna como resultado as entidades negócio que tem relacionamentos entre si [11]. Para que se estabeleçam relacionamentos válidos, cada proprietário de entidade negócio deve aceitar e concordar com o relacionamento [11] o que é natural e fluente entre

empresas do mesmo grupo. Este procedimento previne que uma entidade negócio seja relacionada a outra, sem que ambas tomem conhecimento explícito sobre este fato [11]. Este dispositivo constitui-se mecanismo adequado para modelar o relacionamento hierárquico estruturado entre divisões e departamentos de grandes corporações [11].

Como serviço complementar ao gerenciamento de relacionamento entre entidades negócio, o UDDI encarrega-se de avisar a entidade negócio todas as vezes que outra entidade negócio tenta criar um relacionamento com ela. Podem haver três modalidades de relacionamento : pai-filho que indica uma entidade como pai de outra entidade, par-a-par onde as entidades são consideradas parceiras e o relacionamento identidade informando que as entidades negócio são a mesma organização [11].

3.5.9 – Interface UDDI para SOAP

Com a finalidade de permitir a busca e descoberta de organizações, dos serviços oferecidos por estas organizações e das informações técnicas necessárias para a utilização destes serviços, através da abordagem solicitação/resposta, o UDDI disponibiliza APIs que tornam estas funções executáveis através de programas [11].

Além das APIs para funções de busca e localização de organizações e serviços, UDDI disponibiliza APIs adicionais que permitem também o registro, inclusão de entidades negócio, entidades negócio serviços, templates de ligação e tModels de modo programático [11]. Dois tipos de APIs para interação, via programa, estão disponíveis. O primeiro denomina-se API de publicação a qual é empregada para interação entre aplicações do publicador e o registro UDDI, permitindo a criação, modificação e exclusão de registros completos e dos dados contidos nestes [11]. O segundo tipo de API é denominado de API de busca e localização, destinada a permitir que aplicações realizem pesquisa, busca e localização de informações sobre registros [11].

A API de busca e localização de registro no UDDI disponibiliza três padrões que podem ser empregados para a obtenção de informações : o padrão browse, o padrão drill-down e o padrão invocação. O padrão de API browse permite ao cliente fornecer um critério genérico de busca para obter um conjunto de elementos como resposta. Deste conjunto, seleciona-se um ou dois resultados e aprofunda-se na pesquisa [11]. Neste cenário, inicia-se a pesquisa para verificar-se se um ou mais negócios estão registrados no UDDI. O resultado da pesquisa pode conter informações como as entidades negócio que serão retornadas pela API find_business [11]. Se desejar aprofundar a pesquisa

pode-se solicitar que sejam reunidos todos os serviços disponibilizados pela organização utilizando-se a API find_service que retornará todas as entidades serviço negócio e finalmente para ser obter informações técnicas sobre algum serviço, como por exemplo as versões em uso e a assinatura do mesmo, pode-se empregar a API find_binding para que sejam recuperados todos os tModels referentes ao serviço e suas versões válidas [11].

Para localizar entidade negócio, emprega-se o padrão drill-down que geralmente é usado na interface Web do UDDI, na qual primeiramente se clica em "procurar serviço" tendo-se como retorno parte encontrada do negócio apropriado pelo qual o serviço é oferecido [11]. A interface web permite o drill deeper, localizando entidades negócio a partir do contexto corrente de serviço, como se a navegação fosse no próprio registro [11]. Este padrão possibilita a localização das entidades de negócio a partir de detalhes fornecidos sobre os serviços que as mesmas disponibilizam.

O padrão de invocação baseia-se em uma visão moderna instalada a partir de paradigmas surgidos de Web Services. Um destes é o fato de que clientes podem descobrir dinamicamente Web Services e por isso a presença de camadas cliente e stub não são mais necessárias [11]. Neste cenário a procura por Web Services seria sobre templates de ligação que atendessem aos requisitos do cliente. O padrão de invocação estaria baseado na evolução do padrão Service Locator (divulgado pelo J2EE blueprints), segundo o qual o UDDI manteria em cache templates de ligação de determinados Web Services disponíveis para consulta [11]. O cliente interessado consultaria os templates de ligação disponíveis no UDDI e selecionaria o Web Service mais adequado. Em caso de falha ou de desistência em utilizar o Web Service selecionado, o cliente voltaria ao UDDI e consultaria novamente o UDDI no template de ligação que forneceria novas opções de Web Services dentro dos requisitos recomentados [11]. Chamadas não respondidas por Web Services que estivessem no cache do UDDI, seriam substituídas por Web Services capazes de responder [11]. Como vantagens para este padrão menciona-se a possibilidade de estabelecer mecanismo de monitoramento para taxação e cobrança de uso e o controle de valor da informação de ligação e corretagem de serviços [11].

Uma das vantagens da implementação do UDDI na arquitetura orientada a serviço é prover um ponto único de referência para determinar a localização de um serviço. Quando a informação de ligação do template de ligação precisa ser controlada por outro servidor em cenários de troca de roteamente, recuperação de desastre ou troca de endereço de invocação, pode-se utilizar a estrutura acessPoint, que encontra-se dentro do template de ligação, e que guarda informações para que seja determinado o ponto alternativo de acesso para o Web Services [11]. Com este dispositivo, o cliente do Web Service terá alternativa para localizá-lo em situações de falha do endereço habitual de uso na rede.

3.5.10 – Relacionamento UDDI /SOAP/WSDL

Emprestando as melhores práticas de SOAP e XML, UDDI define o nível que permite as organizações compartilharem mecanismos de pesquisa entre si, descrevendo-se mutuamente [11]. Isto ocorre pelo fato do registro UDDI poder ser acessado via SOAP. Um registro de serviço pode expor qualquer tipo de interface de serviço. UDDI permite para registro de serviço a descrição do tipo de serviço, incluindo formatos como WSDL, ASCII, RosettaNet e outras descrições e abordagens [11]. Idealmente, porém recomenda-se a utilização do WSDL para a descrição [11]. Entretanto a razão primária para armazenar WSDL no UDDI é tornar fácil e rápido para os clientes a descoberta e o acesso ao documento WSDL e opcionalmente a geração programática de proxy para o serviço [11]. Cenários como a implementação de serviço que contem múltiplas documentos de serviço de interface, serviços de interface que referenciam outros serviços de interface e serviços de interface que disponibilizam provedores de serviço [11] são cenários que justificam concretamente o armazenamento do WSDL no UDDI.

Em resumo, o armazenamento de WSDL no próprio UDDI, justifica-se mais firmemente devido as grandes possibilidades fornecidas pela flexibilidade do XML e do elemento import presente na construção do WSDL, que juntos permitem uma grande e complexa gama de combinações possíveis entre as descrições do serviço, as quais seriam gerenciadas com grande dificuldade e alta possibilidade de ocorrência de erros se não contassem com a estrutura única disponível no UDDI, como indicam os exemplos em [11].

3.5.11 – Internacionalização e múltiplos idiomas

Considerando que os registros disponíveis no UDDI serão consultados por usuários do mundo todo, o suporte para alguns idiomas foi adicionado a várias entidades internas da estrutura de informação do UDDI [11]. A letra U da sigla UDDI representa a palavra Universal o que quer significar que o registro necessita oferecer suporte para várias regiões e organizações internacionais que registram e utilizam os serviços disponibilizados [11]. UDDI possui capacidade para classificar, ordenar entidades empregando várias linguagens e idiomas a partir de vários esquemas para a mesma linguagem ou idioma. Adicionalmente, UDDI permite esquemas adicionais de ordenação específico para cada idioma e disponibiliza mecanismos de busca que produzem resultados consistentes de qualquer modo, independente da língua ou idioma em que se esta pesquisando [11].

A possibilidade de oferecer suporte a vários idiomas e línguas do UDDI é fornecida através de API para internacionalização que permite a inserção de descrições em vários idiomas nas entidades negócio [11]. As funções suportadas pela API de internacionalização de UDDI são : descrições e nomes em vários idiomas/língua, vários nomes no mesmo idioma, formato de endereço internacionalizado, comparações dependentes do idioma [11].

Cada idioma/língua suportado pelo registro UDDI é baseado no Unicode 3.0 especificação ISO 10646 que fornecer referência e suporte para a grande maioria dos idiomas e línguas vigentes em uso [11]. Cada idioma suportado possui seu próprio comportamento quando se trata de comparação a fim de produzir ordenação e organização. Este comportamento está relacionado ao fato do idioma basear-se em uma estrutura formativa que pode ser alfabética, silábica ou ideográfica [11].

Idiomas e línguas que compartilham os mesmos conjuntos de sinais alfabéticos como Inglês, Espanhol, Francês e Português, possuem diferentes pesos para comparações e ordenação para as letras, estes pesos estão relacionados às línguas/idiomas nas quais estas letras são usadas [11]. Para línguas/idiomas que possuem letras maiúsculas e minúsculas, ditas bicamerais, a ordenação deve atender a mais uma restrição considerando o fato de aplicar ordenamento levando em conta maiúsculas e minúsculas ou ignorando totalmente a bicameralidade do idioma/língua [11]. Para idiomas/línguas baseadas em ideogramas como o Chinês e outros que possuem um grande conjunto de caracteres, os algoritmos de comparação dependerão da opção a ser adotada que poderá ser por comparação fonética ou ordem dos caracteres dentro do conjunto [11].

Com a globalização, o atributo referente as zonas de tempo tornou-se importante aspecto na comunicação humana. Registros UDDI permitem a publicação de informações sobre contatos para negócios, como telefones e números de fax, com a importante possibilidade de anexar horários adequados para contatos, de modo que os negociadores podem indicar a zona de tempo para contato, especificando esta informação como parte do endereço [11].

Os formatos para endereços são diversificados em várias partes do mundo, UDDI dispõe de mecanismos capazes de suportar estas diversidades [11]. Considerando que em muitas localidades do mundo a especificação de endereços emprega diferentes elementos, como número de lotes, identificação de edifícios, números de andares, subdivisões e outros. No registro UDDI o endereço é suportado pelo elemento adress que faz parte da estrutura de dados entidade negócio. A flexibilidade do endereço encontra-se no fato de que o elemento endereço pode conter elementos adressline os quais podem acomodar os mais diversos formatos [11]. Um registro UDDI pode ter

endereços em vários formatos que são retornados de acordo com a língua/idioma selecionada para a busca da entidade negócio.

3.5.12 – Registros privados de UDDI

Muitas organizações provavelmente não utilizarão Web Services externos ainda durante alguns anos. Inicialmente, empregarão Web Services para possibilitar a integração entre sistemas legados, aplicações de relacionamento com clientes e outros sistemas internos [11]. Neste contexto, faz sentido que a organização considere a implementação de seu UDDI privado como uma das alternativas mais confiáveis no gerenciamento de Web Services que tendem a crescer, mesmo internamente, no que diz respeito as variáveis complexidade e quantidade.

Uma implementação privada de registro UDDI torna-se adequada para organizações que necessitam operar com base em sistemas orientados a serviço [11]. Um vez que sistemas legados venham a ser transformados, empacotados como Web Services e tornem-se orientados a serviço poderão disponibilizar funcionalidade para um grupo específico de outros sistemas que necessitarão ser identificados, autenticados e autorizados e poderão ainda estar localizados na intranet, na Internet ou na rede local privada [11]. A implementação do registro privado produz benefícios como melhoria nos mecanismos de testes, garantia de qualidade dos Web Services, disponibilidade de catálogo de serviço para consumo de usuários internos e mecanismos para busca e localização de serviços na rede privada considerando padrões do próprio segmento de negócios onde a organização atua [11].

Organizações realmente grandes, de escala mundial, e vendedores independentes de software possuem centenas e até mesmo milhares de componentes para aplicações reutilizáveis [11] . Com a criação de um registro UDDI privado, esta classe de organizações podem registrar estes componentes facilitando de modo considerável o compartilhamento interno dos mesmos [11].

O sucesso dos registros privados UDDI ocorreu antes dos registros públicos [11]. No caso dos registros públicos existem vários obstáculos a considerar, pois estes devem aceitar inclusões de registros de vária organizações, garantir qualidade de serviço em vários níveis e desenvolver e encontrar métodos para obter receitas a partir das aplicações registradas [11]. No caso dos registros privados, o controle centralizado é executado por parte do departamento de tecnologia da informação da corporação e não há a necessidade de que o UDDI privado produza receitas, sua principal finalidade é servir aos objetivos da corporação [11].

Benefícios como a restrição de permissão sobre que tipo de usuário pode ter acesso a serviços especiais podem ser obtidos somente em registros privados. Nesses cenários as organizações podem controlar quem pode ou não publicar serviços e autenticar quem está publicando. Sistemas de log e rastreamento podem ser implementados, fazendo com que toda a atividade do registro possa contar com esquemas de auditoria de alta precisão, possibilitando ao administrador do UDDI enumerar e diferenciar todos os clientes que executaram determinado critério find [11] e outros aspectos de uso do mesmo.

Em UDDIs públicos muitas consultas retornam resultados inúteis, muitas vezes por não haver uma categorização taxionômica adequada a verticalização da cadeia produtiva de algumas indústrias [11]. Alguns parceiros de negócios que decidem manter UDDIs privados estabelecem contratos formais para a manutenção de registros internos mais confiáveis e focados em segmentos verticalizados de industrias [11].

Muitas indústrias que optam por esta modalidade de utilizar UDDI, consideram que estarão pesquisando somente entre serviços aprovados e verificados provenientes de parceiros de negócios confiáveis e formalmente registrados que foram aceitos como participantes do UDDI [11]. Neste tipo de associação para cada nova inclusão no UDDI existe aprovação prévia, bem como para cada exclusão do UDDI um destrato formal é realizado [11].

A aplicação de UDDIs privados pode estender-se, entre outros, aos seguintes negócios : mercado de registros, portal de registros e registro para a integração de aplicações corporativas. O mercado de registro pode ser mantido por consórcios de industria que colaboram e competem entre si em uma cadeia vertical de valor. Nesta modalidade muitos negócios estão colaborando para a formação de consórcios que tornam a administração do negócio mais fácil [11]. O arranjo para esta aplicação ocorre da seguinte maneira : um grupo de grande compradores cria um UDDI no qual fornecedores, previamente identificados, são convidados (pagando ou não) a publicar suas ofertas de serviços, permitindo que todos participem competindo mutuamente. A vantagem para os compradores ou tomadores de serviço é que há apenas um lugar a ser pesquisado, padrões preestabelecidos, fornecedores prequalificados e confiáveis e o custo de operação é extremamente reduzido com alto nível de automação [11].

Entre as aplicações possíveis dos UDDIs privados temos a possibilidade de registrar em UDDIs públicos apontadores para UDDIs privados, incluindo estes apontadores em todas as categorizações que podemos considerar, sejam atendidas pelo UDDI privado. Desta forma a parte

"publica" nosso UDDI privado poderá ser encontrada no UDDI público, ficando a critério do cliente adequar-se ou não as exigências de uso apresentadas [11].

Registros internos para a integração de aplicações corporativas são empregados da mesma maneira que registros entre parceiros cadastrados previamente, exceto que estes registros conterão serviços disponíveis apenas de um departamento para outro departamento ou divisão da mesma organização [11]. Esta forma de registro deve ser primariamente empregada por organizações que desejam restringir de forma drástica a criação de novos entidades como tModel, entidade negócio, template de ligação em um grupo centralizado [11]. Este tipo de registro nunca deve ser exposto na Internet para uso externo e deve estar restrito ao uso interno da corporação [11].

Implementar e disponibilizar um registro privado UDDI constitui-se um desafio para que as tecnologias relacionadas ao paradigma de Web Services sejam disponibilizadas em sua completude. Algumas considerações, provenientes da experiência e do amadurecimento da tecnologia de Web Services, se fazem necessárias, quando da intenção de se implementar um registro UDDI privado : a grande maioria dos Web Services não utiliza nenhum tipo de registro público UDDI para divulgar-se; a instalação de um registro privado de UDDI só faz sentido para grandes organizações que estão classificadas na Fortune 500, nas quais o grande número de serviços criados são empregados por toda a organização ou interligando divisões; um UDDI privado faz muito pouco sentido para pequenos arranjos de tecnologia da informação, onde o número de serviços será pequeno e estes serão usados por poucas aplicações; para um pequeno número de serviços destinados a um pequeno número de usuários o mais indicado é publicar o conjunto de WSDL em uma URL específica; ao adotar um registro privado é fundamental que a organização padronize um fornecedor de preferência o mesmo que já atende a outras áreas da empresa em Web Services [11].

3.6 – Outros padrões de suporte a Web Services

A expansão de Web Services como proposta de modelo para computação distribuída no ambiente de Internet[10] e a sua consolidação através do estabelecimento de padrões fundamentais[12], possibilita que iniciativas da indústria de computação e padrões correntes venham a disponibilizar mecanismos de aderência e conformidade com esta nova tecnologia[10].

3.6.1 – Web Services Choreography Interface (WSCI)

O WSCI constitui-se uma iniciativa das empresas Sun Mycrosystems, BEA, Intalio e SAP. Tem a finalidade de definir o fluxo de mensagens trocadas entre processos particulares de Web Services que comunicam-se[10]. O WSCI descreve de modo coletivo o modelo de fluxo de mensagens entre Web Services, possibilitando uma visão global do processo desenvolvido durante a as interações que ocorrem entre os Web Services que se comunicam[10]. Este padrão facilita a integração de Web Services e processos de negócios, permitindo que os primeiros sejam integrados a rotina dos segundos dentro de uma organização e também entre várias organizações[10].

Como pode ser notado em [12], este padrão encontra-se no topo da pilha de descrição de Web Services, juntamente com outros padrões.

3.6.2 – Web Services Flow Language (WSFL)

Constitui-se em uma linguagem baseada em XML, proposta pela empresa IBM, para descrever a composição de Web Services[10]. As composições de Web Services são categorizadas como modelos de fluxo e modelos globais. Modelos de fluxo são usados para especificar modelos de negócios ou fluxos de processos baseados em Web Services. Modelos globais são empregados para definir ligações entre interfaces de Web Services que interagem com operações de interfaces de outros Web Services[10]. O emprego de composições WSFL possibilita a aplicação de ampla coleção de padrões de interação entre Web Services participantes em um processo de negócio, especialmente em interações hierárquicas e relação par-a-par[10]. Como pode ser notado em [12], este constitui-se em um dos padrões que encontra-se no topo da pilha de descrição de Web Services.

3.6.3 – Directory Services Markup Language (DSML)

Através de XML Schema define representação para informação estruturada de diretório como um documento XML, permitindo a publicação e o compartilhamento de informações sobre diretório via protocolos de Internet como HTTP, SMTP e outros[10]. O DSML não define atributos para a estrutura de diretório ou para o acesso a informação. Um documento DSML define as entradas do diretório, o esquema do diretório ou ambos e pode ser usado em compatibilidade com qualquer padrão de diretório da indústria, como por exemplo o LDAP[10]. Em resumo, o DSML define o padrão para a troca de informações entre serviços de diretório diferentes possibilitando a interoperabilidade entre estes[10]. Originalmente proposto pela empresa Bowstreet, como um padrão,

recebendo, mais tarde, a aderência de outras empresas como Sun Microsystem, IBM, Oracle, Microsoft [10].

3.6.4 – XLANG

Tem a mesma finalidade do padrão WSFL. Constitui-se em uma linguagem baseada em XML que tem como objetivo definir fluxos para processos de negócios empregando Web Services. Dispõe de notação para expressar ações e operações complexas com Web Services[10]. Foi desenvolvida pela empresa Microsoft e implementada em um de seus produtos para integrar EIA (Enterprise Aplication Integration) em comunicações B2B[10]. Em [12], este padrão divide o topo da pilha de descrição de Web Services com WSFL.

3.6.5 – Business Transaction Protocol (BTP)

Este padrão disponibiliza especificações para a execução de transações distribuídas em Web Services, permitindo o gerenciamento flexível de transações XA-compatíveis e two-phase commit em motores de transações[10]. Constitui-se uma iniciativa da organização OASIS que facilita a disponibilização, em larga escala, de aplicações B2B disponibilizando confiabilidade para transações distribuídas em Web Services[10].

3.6.6 – XML Encription (XML ENC)

O XML ENC constitui-se em padrão de segurança que está baseado em XML e possibilita a criptografia de dados usando representação XML. Sua aplicação em Web Sevices é garantir a troca segura de dados entre os participantes da interação cliente e provedor de Web Services[10].

3.6.7 – XML Key Management System (XKMS)

Chaves públicas e certificação digital constituem-se padrões de segurança amplamente aceitos na Internet. A integração destes padrões com Web Services é realizada através do XKMS que é baseado em XML e integra a infra-estrutura de chaves públicas (PKI Public Key Infrastructure) e a certificação digital usadas para a segurança de transações na Internet, especialmente empregadas em Web Services[10]. O padrão XKMS constitui-se de duas partes : o X-KISS (XML Key Information

Service Specification) e o X-RSS (XML Key Registration Service Specification). O X-KISS define o protocolo para segurança e confiabilidade que resolve a informação da chave pública contida em elementos X-SIG (XML Signature). A parte X-KRSS descreve como a informação da chave pública é registrada[10].

3.6.8 – XML Signature (XML DSIG)

A criação e a representação de assinaturas digitais em XML são definidas pelo padrão XML DSIG, que baseado em XML, especifica regras de sintaxe e processamento para as mesmas[10]. Na tecnologia Web Services, a utilização de assinaturas digitais em transações baseadas em XML adiciona facilidades como autenticação, integridade de dados e suporte para dados no-repudiation durante a troca de informações, na comunicação entre parceiros[10].

3.6.9 – Extensible Acess Control Markup Language (XACML)

O relacionamento entre parceiros em Web Services deve ser regulado de alguma forma. O XACML constitui-se em um padrão, baseado em XML, para especificação de políticas e regras de acesso a informações sobre recursos disponibilizados na Web. Sua aplicação na tecnologia de Web Services é permitir a especificação do conjunto de regras e permissões sobre recursos compartilhados na comunicação entre parceiros[10]. XACML é uma iniciativa da organização, sem fins lucrativos, OASIS em seu comitê de serviços técnicos de segurança[10].

3.6.10 – Security Assertions Markup Language (SAML)

A troca de informações de autenticação e autorização constitui-se em uma das funções indispensáveis nos modelos de computação distribuída. O padrão SAML baseado em XML, de procedimentos e passos para a troca de autorização e autenticação entre parceiros[10]. SAML utiliza um protocolo genérico, baseado em XML, é composto de no formato solicitação e resposta que pode ser acoplado em muitos modelos de comunicação e protocolos de transporte. Um dos objetivos principais do padrão SAML é disponibilizar o login único para as aplicações participantes do Web Services[10]. SAML é uma iniciativa da organização, sem fins lucrativos, OASIS em seu comitê de serviços técnicos de segurança[10].

3.6.11 – ebXML

Para atender a necessidades inerentes ao B2B (Business to Business), existe esforço concentrado por setores da indústria de estabelecer padrões mais rígidos para registro e repositórios aos quais a tecnologia UDDI ainda não consegue atender [11]. ebXML define um mercado global onde empresas podem encontrar mutuamente, em meio eletrônico, na Internet, e conduzir processos de negócios baseados em colaboração, cooperação e transação. Estes processos baseiam-se em especificações, definidas na Internet, que estabelecem padrão comum para a realização de processos de negócios, modelo de informação de negócios, processo de colaboração entre negócios, perfil para parceiros colaborativos, acordos e mensagens [10].

A iniciativa da tecnologia ebXML é coordenada pelo UN/CEFACT (United Nations Center For Trading Facilitation in Eletronic Business) e OASIS (Organization for the Advanced of Structured Information Standards) [10]. Segmentos populares e verticalizados de atividades industriais como OTA (Open Travel Alliance), OAGI (Open Application Group, Inc.), GCI (Global Commerce Initiative), HL 7 (Health Level 7, padrão para organizações de saúde) e RosetaNet (um dos padrões do comitê XML) endossaram esta iniciativa [10].

No contexto do modelo de Web Services, ebXML disponibiliza modo bem definido e formal de procedimentos para comércio eletrônico e processos de B2B comunicarem-se e interagirem definindo padrões para processos de negócios, perfil de parceiros, acordos, serviços de registro e repositório, serviço de mensagens e componentes do núcleo [10]. Completando e estendendo componentes básicos de Web Services como SOAP, WSDL e UDDI.

Entre as extensões oferecidas pela tecnologia ebXML temos : a possibilidade de especificar processos de negócios através do BPSS (Business Process Service Specification); ebXML CPP (Collaboration Protocol Profile)/CPA (Collaboration Protocol Agreement)possibilitam que o perfil dos parceiros de negócio e os acordos sejam definidos, produzindo assim coreografia de transações de negócios; serviço de mensagem ebXML agrega valor ao SOAP transportando, roteando e empacotando mensagens com confiabilidade e segurança; o registro ebXML define serviços de registro, protocolos de interação, definições de mensagens e o repositório ebXML comporta-se como acumulador de mensagens para informações compartilhadas.

O provedor de registro ebXML troca informações com os demais registros ebXML, formando uma federação, os quais podem ser facilmente encontrados pelos UDDIs. Esta facilidade permite que o UDDI encontre uma lista de negócios registrados em todos os ebXML apenas

consultando um deles que se comporta como registro/repositórios[10]. Os componentes do núcleo ebXML disponibilizam catálogo contendo componentes de processos de negócios que instruem sobre funcionalidades comuns empregadas na comunidade de negócio na qual o ebXML registro/repositório funciona. Estes componentes estão relacionados com segmentos como pagamentos, cotação de preços, inventários e outras áreas [10].

ebXML representam uma evolução em relação a UDDI em vários aspectos [11]. São descritos por WSDL e expostos como Web Services [11]. Sua adoção em massa, apesar de sua demonstrada superioridade e poder depende basicamente de dois fatores : político, pois alguns dos membros do OASIS não suportam as tecnologias exigidas para a sua implementação e funcionais, pois a sua implementação ficou por conta de vendedores e implementadores de software [11].

3.7 – Conclusão

Nesse capítulos discorremos sobre as três principais tecnologias envolvidas no núcleo de Web Services : SOAP; WSDL e UDDI. Bem como a base de todas elas a linguagem de marcação XML e suas principais facilidades : Namespaces; DTD e XML Schemas. Vale ressaltar que de modo geral esta tecnologias podem ser consideradas "naturais" da Internet pois estão em harmonia com padrões já estabelecidos como HTTP, TCP/IP e o próprio HTML. Assim, estas tecnologias podem ser aplicadas sobre qualquer plataforma, processador, sistema operacional ou linguagem de programação que seja capaz de conectar-se a Internet.

Agregando valor as tecnologias do núcleo de Web Services apresentamos, de modo resumido, padrões como ebXML, SAML, XKMS e outros, que estendem a aplicação dessa tecnologia para que atenda a requisitos mais rígidos de uso na indústria de computação distribuída.

Em seguida, passamos a discorrer como a linguagem Java adere aos padrões do núcleo da tecnologia Web Services. Descreveremos que APIs e demais artefatos desempenham funções destinadas a suprir áreas cobertas por SOAP, WSDL e UDDI.

Capítulo 4 – Java e Web Services

4.1 – Introdução

O principal aspecto a ser observado na tecnologia de Web Services é o conceito da interoperabilidade, que deve permitir a qualquer computador, programa, plataforma de hardware ou sistema operacional, que baseados nos padrões estabelecidos na Internet como HTTP, SOAP e XML possa acessar e, se for conveniente, utilizar o Web Services disponibilizado. Para integrar-se a este novo aspecto da interoperabilidade a tecnologia Java disponibiliza conjunto de mecanismos em forma de API que inicialmente distintos tendem, com o amadurecimento da tecnologia de Web Services e da própria tecnologia Java, a aglutinarem-se em conjuntos e ambientes como J2EE.

Web Services tem como tecnologia de base a linguagem XML. Java disponibiliza a API JAX P (Java API for XML Processing) para o processamento e gerenciamento de documentos em XML no que se refere ao parsing destes conteúdos que podem ocorrer de duas formas : SAX e DOM. Esta API tem seu emprego nos extremos do processo de troca de informações ao transformar-se a informação Java em XML para enviar e ao receber a informação XML e transformá-la para Java.

Para a implementação de Web Services em Java no estilo RPC a tecnologia Java disponibiliza a API JAX RPC (Java API for XML RPC) que implementa mecanismos para o desenvolvimento de Web Services que podem ser acessados por qualquer participante da Internet aderente aos padrões estabelecidos, dentro dos modelos de comunicação solicitação/resposta e com base em na tecnologia de containers, porém de modo transparente para o potencial cliente que deseja apenas utilizar o Web Services disponibilizado.

A comunicação orientada a mensagens, também pode ser disponibilizada como modalidade da tecnologia de Web Services. Em Java temos a API JAX M (Java API for XML Messaging) que disponibiliza meios para a implementação e disponibilização de Web Services

orientados a mensagens que podem funcionar em duas categorias de comunicação : síncrona e assíncrona. Porém, em ambas, todas as características da universalidade dos Web Services estão garantidas pois temos a troca de mensagens baseadas em SOAP, HTTP e XML.

Complementando o acrônimo, já mencionado, WUST (WSDL-UDDI-SOAP Tecnologies), Java oferece proposta de padronização para registros de Web Services, os UDDIs, de modo a permitir a manipulação, manutenção e principalmente, a pesquisa e a busca por serviços dentro de padrões que possibilitem a localização simplificada de Web Services por clientes em potencial com qualquer nível de entendimento da tecnologia de Web Services. A proposta de Java está representada na API JAX R (Java API for XML Registries) que traduz em termos concretos padrões para a interação com dois dos principais padrões de registros estabelecidos no mercado e oferece a opção de expansão e implementação da interface para casos particulares que não se adeguem aos padrões oferecidos.

Considerando as características da tecnologia de Web Services, oferecemos a seguir aspectos, sob a perspectiva da tecnologia Java, inerentes aos pontos a serem considerados no desenvolvimento e disponibilização de Web Services empregando-se princípios, conceitos e artefatos disponibilizados por esta tecnologia.

4.2 – JAX P

Tendo como base principal o processamento de XML, a tecnologia de Web Services necessita contar com vários componentes adequados a manipulação desta linguagem que constitui-se a base fundamental para a neutralidade, independência de plataforma e interoperabilidade. O procedimento segundo o qual o XML é processado e interpretado é denominado parser[10], neste processo podemos extrair informações em XML e apresentá-las de outras formas, também é possível validar e verificar a boa formação de um documento XML no processo de parser.

Além do processo de paser do XML, podemos ainda aplicar o processo de transformação, no qual pode-se aplicar vários modelos (templates) ao documento XML e produzir-se diferentes formas de saída[10]. O segmento, o ramo da tecnologia Java encarregado de conduzir diretrizes, oferecer modelos de referência e orientar padrões neste para estas tarefas é denominada JAX P (Java API for XML Processing).

Desde o início e a adoção efetiva do XML como linguagem neutra para a troca de dados na Internet, a XML parsing foi padronizado em dois modos distintos de processamento : SAX (simple API for XML) e DOM (document object model)[10]. Estas duas formas atendem diferentes necessidades referentes ao parsing do XML e produziram APIs adequadas a cada uma delas que permitem ao desenvolvedor emprega-las de acordo com as características do problema de parsing a ser resolvido[10].

Muitos vendedores implementaram soluções proprietárias em APIs Java para ambos padrões de parser, oferecendo opções para modos SAX e DOM[10]. Entretanto, este fato produziu situação complexa para a padronização de APIs, na qual aplicações utilizando parsers de um fornecedor específico, eram obrigadas a empregar chamadas a métodos proprietários para realizar tarefas equivalentes, que em parsers de outros fornecedores tinham métodos completamente diferentes[10]. A maior dificuldade neste cenário era encontrada nos casos para os quais o parser necessitava ser trocado por algum motivo. Nestas situações a aplicação tinha que ser completamente rescrita para adaptar-se ao novo parser, completamente diferente ao anterior[10].

A solução deste problema ocorreu em 1999, com a interseção da Sun Microsystens ao apresentar versão rascunho das especificação para padronização de JAX P 1.0[10]. O objetivo da especificação era oferecer padrão de referência em alto nível de abstração para API JAX P 1.0, [10] deixando detalhes do nível de implementação para desenvolvedores de parsers XML. A primeira versão JAX P 1.0 foi entregue com especificações para suportar parsers SAX 1.0 e DOM nível 1[10]. O foco da primeira especificação apresentada foi centralizado nos parsers não apresentando menção aos procedimentos de transformação que XML já podia sofrer e produzir saídas em formatos distintos. Em seguida a especificação para JAX P 1.1 foi divulgada com suporte e padronização para parsers SAX 2.0, SAX 2 extensões e DOM nível 2[10]. Esta versão também incluía suporte e especificações para transformações XML baseadas no XSLT (extensible stylesheet language transformation)[10]. O propósito por trás da padronização obtida com JAX P é permitir a abstração[10], o não conhecimento, o emprego do conceito de componente, de caixa preta, no que se refere ao parsers que se está utilizando seja ele SAX ou DOM, o parser deve parecer o mesmo, possuir o mesmo comportamento para a aplicação que o emprega. Este tipo de abstração é denominado, conhecido e referenciado, na literatura Java, como interface plugável[10].

4.2.1 – Arquitetura de JAX P

O caráter plugável de JAX P adiciona um nível de abstração ao desenvolvimento e permite aos construtores de aplicações trocar o parser sem que esta atitude venha a afetar a aplicação em sua lógica de negócio[10]. Um parser JAX P pode ser trocado por outro parser JAX P com pouco esforço[10]. A padronização da interface JAX P disponibiliza conjunto de interfaces padrões que encapsulam detalhes por traz da interação com os parsers[10]. Estas interfaces agem como abstrações que preservam, livram, o desenvolvedor de ter que trabalhar diretamente manipulando XML puro[10]. Estas abstrações constituem-se na implementação concreta de parsers SAX e DOM que obedecem padrões rigorosos de interface definidos pelo JAX P, isto se aplica também aos transformadores XSLT construídos segundo os mesmos rigores dos padrões estabelecidos[10].

Entretanto o mais importante sobre a API JAX P é o fato da mesma constituísse em uma camada de abstração [11]. Em momento algum a API JAX P adicionou qualquer componente SAX, DOM, XSLT ou mecanismo para a manipulação e o processamento de XML [11]. Com esta interface criou-se o padrão para que desenvolvedores independentes pudessem oferecer soluções independentes e competitivas de parsers e transformadores XSLT das quais não sem tem controle e nem conhecimento sobre detalhes na camada de implementação. Estas soluções podem ser livremente escolhidas e empregadas pelo desenvolvedor Java que necessita processar XML [11].

A vantagem da padronização em relação as soluções particulares é a sensível redução na curva de aprendizado. JAX P tem por constituição e projeto grande simplicidade e uma vez entendido e aprendido seu funcionamento, a componente plugado sob ela torna-se indiferente ao desenvolvedor. Na Figura 4.2.1-1 apresentamos o esquema de arquitetura lógica da interface JAX P (adaptada de [11]) que demonstra como podem ser plugados parsers de desenvolvedores independentes e de forma natural com ênfase na camada de abstração criada por JAX P. Na implementação de referência fornecida pela Sun MicroSystems, acompanha, apenas como facilidade para experimento, o parser e o transformador XSLT da Apache Fundation, que não fazem parte da especificação [11]. Aprofundando um pouco mais em detalhamento, a Figura 4.2.1-2 apresenta os principais componentes internos da camada de abstração da API JAX P.

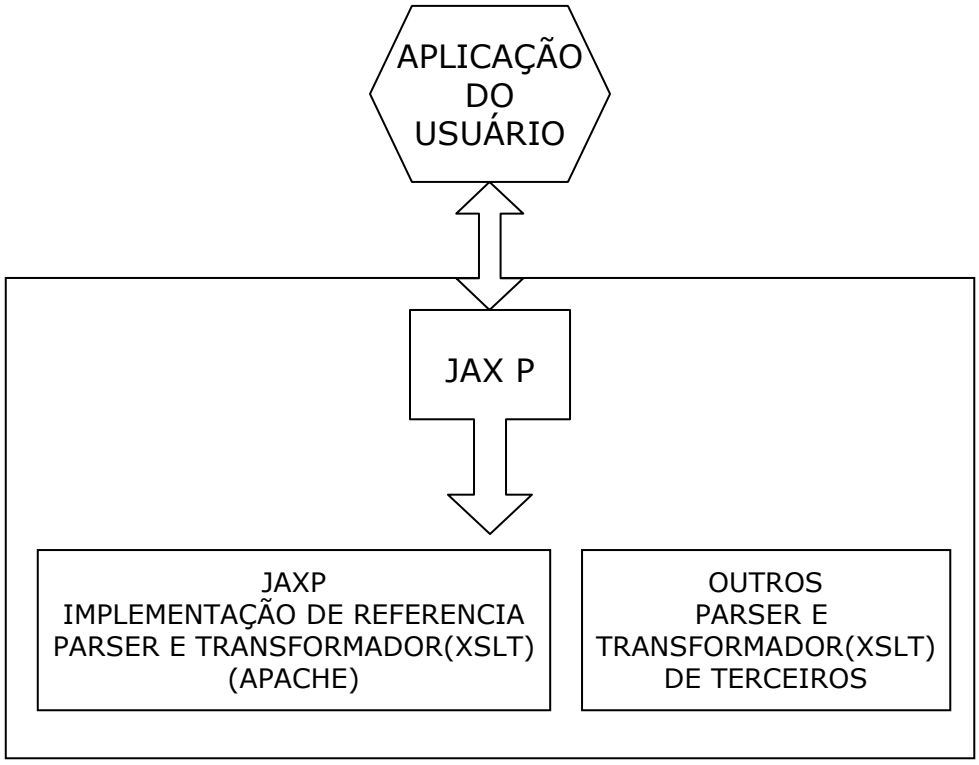


Fig. 4.2.1-1 arquitetura lógica da API JAX P.

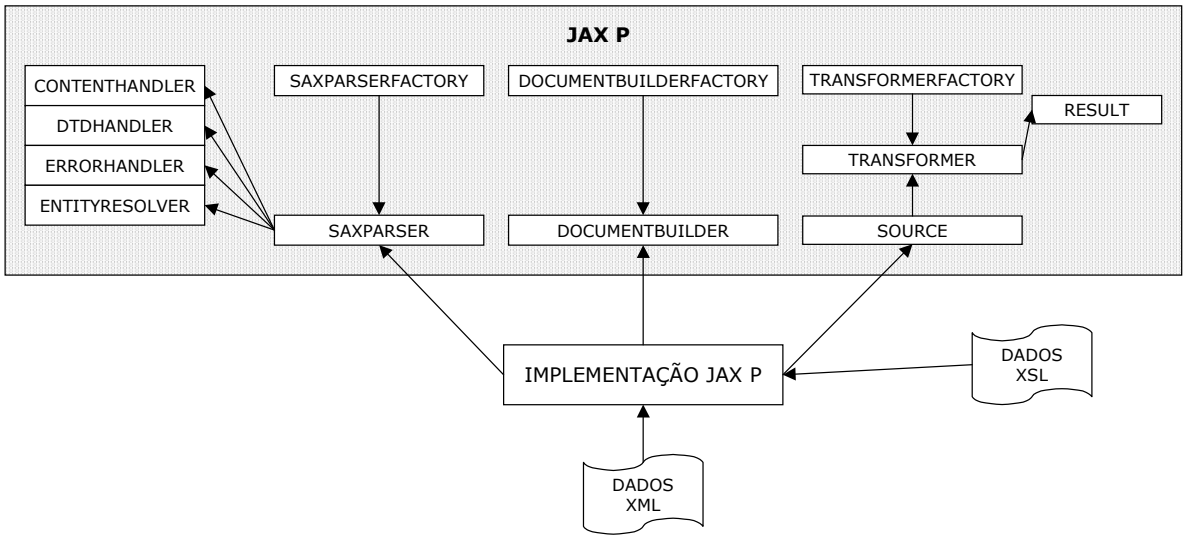


Fig. 4.2.1-2 arquitetura da API JAX P (principais detalhes internos).

4.2.2 – Modelo da API JAX P

Concebido por projeto para tornar-se simples de entender e empregar, o modelo API JAX P, constitui-se de classes e interfaces reunidas sob o pacote Javax.xml.parsers do qual estão no topo as API para SAX e DOM[10]. As classes e interfaces responsáveis pelo suporte a transformação

estão localizadas em Javax.xml.transform que disponibilizam utilitários para transformações e operações em XSLT[10].

Torna-se importante ressaltar que a API JAX P encontra-se disponível no site da Sun com o que chama-se de parser de Crimson que constitui-se o parser de referência de implementação JAX P[10] ou seja é uma implementação que demonstra como os outros parsers, de fornecedores particulares, devem funcionar em relação a API JAX P. Mesmo sendo fornecido com a API JAX P no pacote com.sun.xml.parser este componente não faz parte da especificação formal da API JAX P. O uso desde componente está além, não integra o conceito em alto nível criado para a interface que possibilita o uso e o acesso direto e plugável de parsers[10].

O conceito fundamental adotado para o modelo API JAX P é o padrão de factory (fábrica)[10]. Este padrão possibilita a instanciação transparente da implementação do parser[10] seja ele qual for. Na realidade este é o mecanismo que promove a abordagem da plugabilidade (o encaixe predefinido, formalizado, o acordo de funcionalidade conjunta) possibilitando que desenvolvedores definam a implementadores definam a implementação de factories como parâmetros que serão passados para a maquina virtual Java[10].

4.2.3 – Implementação JAX P

Com a adoção do padrão para a implementação de parser XML e transformadores XSLT, atualmente existem muitas implementações, entre estas Sun Crimson, Apache Xerces, IBM XML parser, Oracle e outros[10]. Entre as principais e compatíveis com a versão JAX P 1.1 podemos mencionar Apache Xerces 2 que constitui-se parser XML e Xalan 2 transformador XSLT[10].

O estabelecimento da interface JAX P fez com que a mesma fosse incluída na versão 1.4 do kit básico para desenvolvimento Java, JDK (Java Development Kit) como referência de implementação. O conjunto inclui as implementações da Sun Crimson para parsers XML e Apache Xalan para transformadores XSLT[10]. Estas implementações de referências estão disponíveis para serem utilizadas, bem como outros parsers e trasnformadores XSLT devem ser plugados para que a eficiência da interface seja testada bem como a dos novos componentes.

4.2.4 – Processamento XML com SAX

O processamento de XML através da API SAX (que esta sob API JAX P), está baseado em modelo de eventos que ocorrem no processamento, quando elementos de dados do documento XML são interpretados em base seqüencial, são produzidas chamadas específicas para construtores padronizados e selecionados[10]. Este comportamento é bastante similar ao do modelo adotado no AWT (Abstract Windowing Toolkit) 1.1 delegação de evento, onde componentes da interface do usuário produziam eventos baseados em estímulos e entradas e estes eventos eram direcionados para construtores padronizados e selecionados que realizavam ações correspondentes ao disparo destes eentos[10].

A principal característica do SAX e sua vantagem considerada mais importante, constitui-se no fato de que em seu modo de operação o mesmo não ler, armazenando completamente, nenhum documento XML para a memória. Devido a este comportamento, considera-se o SAX interface muito rápida e de leve processamento, exigindo baixo e pouco esforço computacional[10]. SAX suporta validação de documentos XML, porém de modo opcional, não tornando obrigatória a consistência de validação para a sua utilização[10]. Como mecanismos de validação um documento XML pode ser checado quanto a conformidade pelo SAX, empregando-se documento de validação de esquema que poderá ser DTD ou XML Schema[10]. SAX realiza, por natureza de projeto, a leitura seqüencial de documentos e não permite acesso aleatório a elementos do documento XML[10].

Existem várias formas de processamento para documentos XML empregando-se a API SAX. Uma das formas é empregando-se a API JAX P (jvax.xml.parsers.*) a qual abstraí muitos detalhes de baixo nível que poderiam produzir eventos e chamadas mais detalhadas para construtores SAX[10], neste caso temos modo de utilização de mais alta granularidade, tratando e produzindo eventos somente para elementos mais expressivos e relevantes do documento XML em questão.

A outra maneira de utilizar-se diretamente SAX seria pela API SAX (org.xml.sax.*)[10] . O uso desta API de modo direto representa mais dificuldade devido aos requisitos que a mesma apresenta, necessitando de maiores detalhes e maior quantidade de passos para a obtenção de resultados muito próximos e equivalentes ao do uso anterior[10]. Entretanto, devido a este nível de detalhamento, a quantidade de eventos e a granularidade para os mesmos pode ser reduzida, gerando-se eventos para todo e qualquer nível do documento XML, isto é, pode-se trabalhar com grão mais finos de elementos XML no documento.

O modelo de processamento SAX pode se considerado bastante simples e enumerado em apenas três passos distintos, explicados a seguir :

- implementa-se uma classe que estende a classe DefaultHandler e processa as chamadas de métodos para todos os tipos de elementos e construtores encontrado no documento XML que compõem as necessidades da implementação em questão;
- instancia-se um nova classe parser SAX. O parser ler o arquivo fonte XML e dispara chamadas e gatilhos implementados na classe DefaultHandler;
- lendo-se a fonte XML seqüencialmente. Com a leitura seqüencial, não é possível o acesso aleatório ao elementos da estrutura do documento. Outras funcionalidades e o atendimento de requisitos mais sofisticados dependem inteiramente da implementação particular dos manipuladores das classes implementadas pela aplicação que emprega o SAX[10].

A simplificidade do modelo de processamento SAX poderá ser observada na Figura 4.2.4-3 – JAX P empregando o modelo de processamento SAX, a seguir, que apresenta, de modo simplificado, este funcionamento. Observe que ao encontrar os elementos no documento XML é produzido um evento, uma chamada, para os manipuladores de evento na classe DefaultHandler, nesta classe, então o desenvolvedor poderá realizar todo o trabalho necessário ao atendimento dos requisitos de sua aplicação.

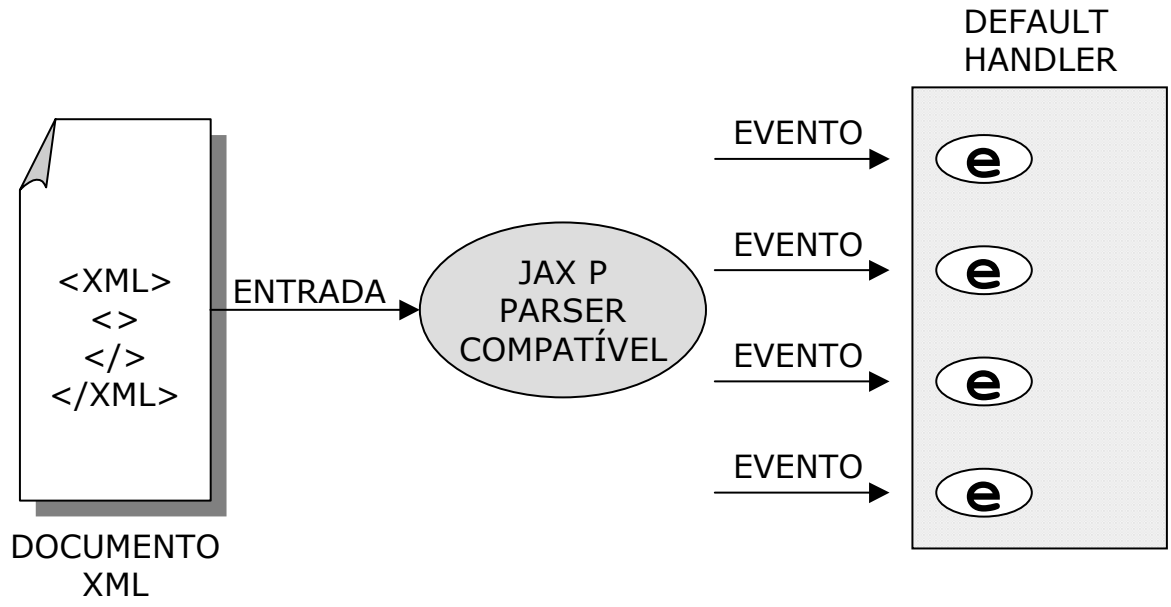


Fig. 4.2.4-3 – JAX P empregando o modelo de processamento SAX.

4.2.5 – Processamento XML com DOM

Definido e mantido pelo W3C o modelo de processamento de documentos XML DOM possui a seguinte premissa " o Modelo de Objeto para Documento XML é neutro no que refere a plataforma e linguagem para interfaces, o que deverá permitir que programas e scripts acessem dinamicamente e atualizem seu conteúdo, estrutura e estilo de documentos que adotem este padrão."[10].

O modelo de processamento DOM consiste em ler, carregar inteiramente documento XML para dentro da memória, construindo uma árvore semântica que representa a estrutura de dados do documento[10]. Este processo requer substancial esforço computacional de processamento e quantidade de espaço em memória quando se trata de documentos XML longos e/ou complexos[10]. A montagem e a leitura de todo o documento XML para dentro da memória exige, além dos recursos computacionais e de memória, tempo para ocorrer e ainda pode tornar o processamento do documento lento, em sua manipulação, dependendo do recurso de hardware do qual se disponha.

Entretanto, o que poderia se tornar uma desvantagem, em algumas situações e aplicações, torna-se necessário, indispensável e uma vantagem. Com todo o documento XML carregado e disponível na memória, a API DOM introduz, disponibiliza a possibilidade de manipulação, de navegação pelos dados e estrutura XML, permitindo operações como a inserção de elementos, a exclusão de elementos e a alteração de elementos que compõe a árvore semântica estrutural do documento XML[10]. Diferindo fortemente da API SAX, DOM permite, de modo natural, o acesso aleatório a qualquer um dos nós, dos elementos da árvore[10]. É permitido o emprego de mecanismo para validação de documentos XML na API DOM que oferece as opções de utilização DTD e XML Schemas como esquemas de suporte a validação, entretanto de modo opcional não obrigatórios[10] pode-se empregar a API DOM sobre documentos XML sem no entanto validá-los.

Comparativamente, o modelo de processamento da API DOM demonstra-se muito mais complexo que o modelo empregado ela API SAX [10]. Considerado mais intensivo no consumo de recursos que o modelo da API SAX, por ler e carregar todo o documento XML na memória[10]. A seguir passamos a oferecer três procedimentos básicos para a implementação do modelo de processamento empregado pela API DOM :

- deve-se instanciar uma nova classe Builder. A classe Builder é responsável por ler o dado XML e transformá-lo na arvore semântica estrutural que representa do documento XML lido;

- cria-se o objeto documento no qual o dado XML foi transformado e poderá ser manipulado;
- utiliza-se o objeto documento para acessar os nós representando os elementos do documento XML[10].

A fonte XML é lida inteiramente na memória e representada pelo objeto documento. Isto possibilita à aplicação, acesso a qualquer nó de modo aleatório, esta possibilidade não está disponível na API SAX[10]. Deve-se entretanto considerar que para documentos de tamanho considerado grande a manipulação da árvore poderá representar lentidão e dificuldades devido ao pouco espaço e da complexidade da operação que se desejará realizar sobre os elementos. A Figura 4.2.5-4 JAX P empregando modelo de processamento API DOM mostra graficamente como esta interface funciona e permite notar a forma com a arvore ocupa a memória.

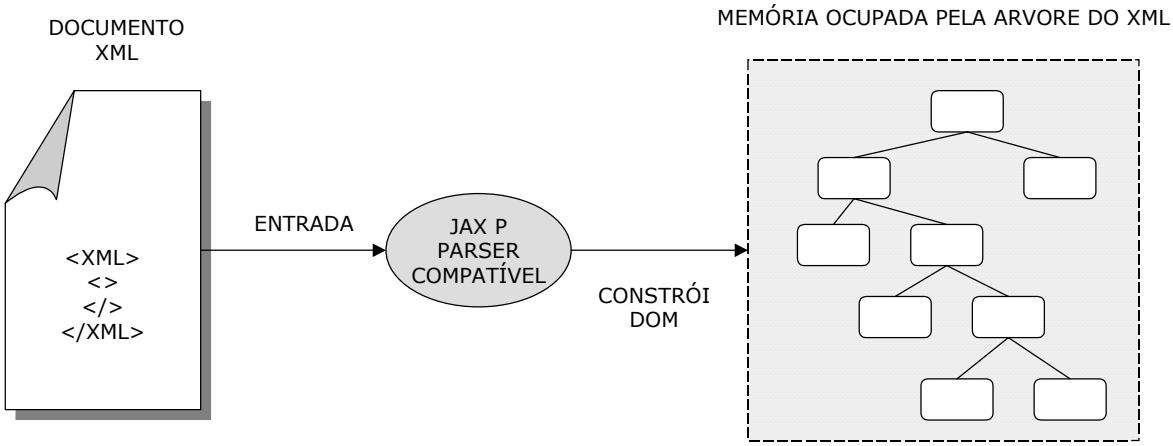


Fig. 4.2.5-4 JAX P empregando modelo de processamento API DOM.

4.2.6 – Processamento XML com XSLT

Para que seja possível a transformação de documentos XML em vários formatos diferentes de saída, a partir da lingugem XSL (Extensible Stylesheet Language), a API JAX P disponibiliza interfaces para transformadores XSLT que constituem-se em processos capazes de transformar um documento XML de entrada em outro documento de saída baseado em informações fornecidas por um arquivo ou esquema XSL[10].

O entendimento da função de transformação do XSLT em relação as entradas XML deve ser compreendido como processo padrão utilizando XSL para documentos baseados em XML que serão transformados[10]. Este é o processo pelo qual cada template XSL é aplicado a documentos XML, de modo a criar novos documentos nos formatos desejados como por exemplo PDF, XML,

HTML e WML[10]. XSL disponibiliza a sintaxe e a semântica para especificação da formatação, então o processador XSLT realiza a tarefa de formatação descrita no template XSL[10].

A proposição do emprego de XSLT é sempre o seu emprego para a produção de vários formatos de saída para aplicações que possibilitam acesso de tipos de clientes bastante heterogêneos como web browsers, telefones celulares e aplicações Java[10]. XSLT também encontra grande quantidade de aplicações na transformação de formatos XML em outros formatos XML, isto ocorre tipicamente em ambientes B2B[10].

JAX P suporta XSLT de modo que seja possível a incorporação de transformadores XSLT para XML, tornando aplicações desenvolvidas com esta interface neutras em relação ao fornecedor do componente responsável pela transformação de XML em outras saídas empregando o transformador XSLT[10].

No caso dos transformadores XSLT para documentos XML, o modelo de processamento consiste em várias regras definidas no template que devem ser seguidas pelo XSLT[10]. As regras são definidas a partir da combinação do template que funciona como um modelo e um padrão para o documento. A saída a ser obtida deve ser resultado do documento de entrada XML que sofreu o processamento descrito em XSL e executado pelo XSLT[10] . De modo genéricos, devemos considerar que um nó do documento XML é processado por uma regra do template que encontra um padrão preestabelecido. Uma vez localizado o template o mesmo é instanciado e o resultado é criado a partir dele. Este processo continua até que toda a árvore do documento XML seja percorrida recursivamente através de seus dados. A medida que todos os nós são processados, o resultado é conhecido como lista de nós que por sua vez é concatenado com todos os demais resultados[10]. A concatenação e o controle de nós processados e formatados é realizado a partir do XPath que consitui-se em mecanismo de localização e navegação para documentos XML[10].

Obedecendo o padrão da API JAX P, existem três procedimentos básicos necessários para que seja possível a transformação de documento XML em uma saída com formato diferente, processado por um transformador XSLT a partir de um template descrito em XSL, como a seguir listamos :

- obtém-se a classe fábrica de transformação, empregada para instanciar a classe do transformador XSLT. Este passo é muito mais complexo que os passos equivalentes no caso das API SAX e DOM;

- cria-se a classe do transformador XSLT que deve ser informada sobre o formato que será dado a entrada XML ou seja informa-se qual será o template XSL a ser aplicado na entrada para se obter a saída no formato desejado;
- emprega-se a classe transformadora XSLT que já possui o formato XSL, para produzir a saída desejada, a partir da entrada do documento XML. O resultado desejado será enviado no formato programado[10].

A folha de estilo e formato, a XSL é definida separadamente em arquivo e possui a extensão *.xsd. Este arquivo, contendo a folha de estilo deve ser passado no ato da instanciação do transformador XSLT. A Figura 4.2.6-5 emprego do JAX P para transformação XSLT, demonstra, de modo simplificado, o uso desta API JAX P para a produção de saídas em vários formatos, destinados a clientes heterogêneos, a partir de entrada XML.

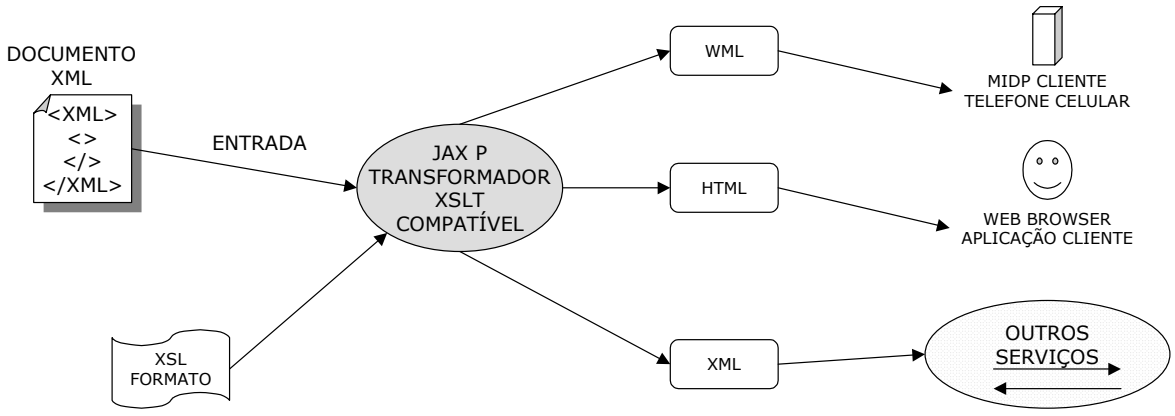


Fig. 4.2.6-5 emprego do JAX P para transformação XSLT.

4.2.7 – JAX B Arquitetura Java para ligação com XML

O emprego em massa da tecnologia XML e suas possíveis extensões permitem ao desenvolvimento de aplicações, a consideração de possibilidades da geração de modos de produzir objetos Java, baseados em definições XML e, de maneira inversa, produzir definições XML a partir de objetos Java[10]. A arquitetura Java para ligação com XML, abreviado de JAX B (Java Architecture for XML Binding), formalmente conduzida pelo JCP (Java Community Process), com o nome de projeto Adelard, constitui-se em especificação de alto nível que define a abstração para a ligação semântica, via classes e interfaces, de objetos Java a definições em XML[10]. A ligação entre classes e interfaces em Java a dados em XML possibilita um modo para que aplicações trabalhem com árvores de dados XML complexos da mesma forma como trabalham com objetos em Java[10].

A conversão de XML para Java pode ser alcançada com o emprego de parsing (JAX P) e a partir desse ponto construir-se objetos Java[10]. A intenção com a adoção do padrão JAX B é possibilitar o uso facilitado e aumentar o poder de processamento[10] da linguagem Java em sua interação com XML.

Com a adoção do padrão JAX B, evitaria o trabalho de desenvolvedores em produzir soluções proprietárias e do esforço desnecessário para manter em funcionamento projetos nem sempre otimizados e adequados[10]. Uma especificação padrão de ligação entre XML Java e Java XML faria com que toda a comunidade de desenvolvedores adotassem técnicas e procedimentos de conhecimento e domínio de todos e surgirão, efetivamente, parsers capazes de gerar automaticamente classes a partir de XML e vice versa[10]. O padrão JAX B possibilitaria um modo para validar documentos definidos por XML Schema e tipos de dados mapeados para XML, forçosamente corretos[10].

A versão corrente do JAX B disponibilizada pela Sun MicroSystems ainda apresenta algumas limitações[10]. Estas limitações ainda representam significativo impacto nas necessidades da indústria para adotá-la em esquema de produção de software. Existem iniciativas de grupos de desenvolvedores [18], [19] que avançam no sentido de superar estas limitações[10]. Na Figura 4.2.7-6 ciclo de vida para ligação Java XML Java (proposta JAX B), apresentamos a idéia básica do que vem a ser esta arquitetura.

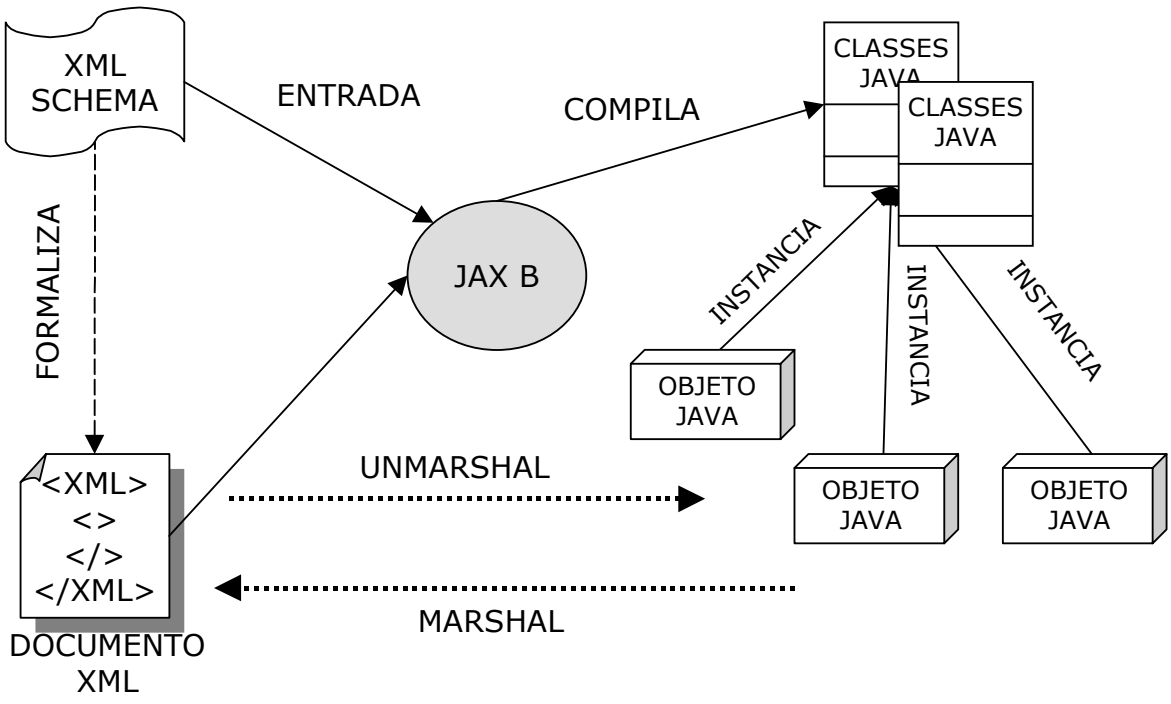


Fig. 4.2.7-6 ciclo de vida para ligação Java XML Java (proposta JAX B).

4.3 – JAX RPC

A API JAX RPC (Java API para RPC) possibilita o desenvolvimento de Web Services em Java incorporando a base XML, na tecnologia de chamada remota de procedures ou RPC (remote procedure call) [10]. Na tecnologia Java para o ambiente de Web Services JAX RPC possibilita funcionalidade baseada em XML para RPC entre o solicitante do serviço (o cliente) e o provedor do serviço [10].

O padrão imposto pela API JAX RPC inclui mecanismos para a criação de Web Services baseados em RPC, bem como ambiente de execução para serviços de aplicações Web Services [10]. Uma das principais funções destas aplicações, é a disponibilização do serviço de solicitação que um cliente pode invocar remotamente através de uma chamada de acesso a um, entre muitos, serviços expostos por um provedor de serviço [10].

Constituindo-se o WSDL em um dos principais componentes para a manutenção do alto grau de interoperabilidade da tecnologia Web Services, o JAX RPC também é responsável, no universo Java, pelo mapeamento entre serviços baseados em WSDL, para classes Java [10]. Isto é JAX RPC pode, a grosso modo, ler WSDL e produzir as classes Java adequadas ao acesso e a execução (a parte de interface, esqueleto, casca) do Web Service descrito pelo WSDL. O retorno também é verdadeiro, pois JAX RPC possui a capacidade de tomar classes de Web Services em Java e produzir o seu respectivo WSDL [10], para que outras linguagens, em plataformas e sistemas operacionais diferentes possam utilizar e acessar os Web Services implementados em Java, com JAX RPC. Isto permite aos desenvolvedores criar clientes JAX RPC para Web Services que permitam a ocorrência de interoperabilidade entre aplicações Web Services escritas utilizando-se outras linguagens e/ou funcionando em plataformas heterogêneas [10].

O mecanismo conhecido em Java como RMI (Remote Method Invocation) empresta bastante semelhança a JAX RPC que difere muito pouco deste, pois o RPC constitui-se em um mecanismo que transporta objetos Java serializáveis entre aplicações Java em ambiente de computação distribuída [10]. Entretanto, JAX RPC comporta-se de forma semelhante só que emprega uma espécie de RPC baseada em SOAP e um mecanismo WSDL para descrever como deve-se invocar Web Services funcionando em ambientes heterogêneos [10]. O mais importante é o fato de que JAX RPC

esconde toda a complexidade da operação na mensagem a ser transmitida sob o empacotamento SOAP possibilitando o mapeamento seguro e adequado entre Java e XML com o apoio do WSDL [10].

Entre as principais razões que tornam o desenvolvimento de aplicações Java Web Services baseados em JAX RPC, exigindo relativo pouco esforço e conhecimento de pouca complexidade sobre SOAP RPC [10] é o fato da especificação JAX RPC ser completamente compatível com a especificação SOAP 1.1 e WSDL possuir uma especificação e suporte adequado e aderente a HTTP elegendo-o como protocolo de transporte primário e default [10].

No contexto de negócio, a implementação de Web Services empregando o padrão JAX RPC para a disponibilização de serviços que possam solicitar e responder utilizando mensagens baseadas em SOAP, trocando conteúdos XML, com parâmetros ou valores de retorno, apresenta-se como tendência com grandes possibilidades de afirmar-se sobre boa parte do mercado de provedores de Web Services[10].

A iniciativa de desenvolvimento do JAX RPC é da empresa Sun MicroSystems como parte da JCP (Java Community Process), apoiada por vendedores de J2EE e demais plataformas baseadas em Java Web Services [10].

4.3.1 – JAX RPC em Web Services

No ambiente de Web Services, JAX RPC define uma API o modo, a forma, os procedimentos que devem ser executados e o ambiente necessário e adequado para que Web Services sejam criados e executados baseados em XML e na chamada remota de procedures (procedimentos) [10]. O solicitante do Web Services invoca o provedor do serviço, solicitando o método que deseja e passando os parâmetros, então recebe como retorno os valores com base em XML usado na solicitação e na resposta para codificar as informações trocadas[10]. Tipicamente, o ponto de acesso do Web Services e a aplicação participante como cliente empregam JAX RPC para a definição e execução de Web Services baseados em RPC [10]. Porém, este comportamento contraria o maior dos objetivos da tecnologia Web Services que constitui-se a tentativa de fazer com que provedores de Web Services e clientes sejam das mais diversificadas e diferentes origens possíveis tendo como único elo de ligação o meio de comunicação a Internet, o HTTP, o SOAP e o WSDL.

Definida para permitir na linguagem Java a implementação de Web Services no estilo RPC, a API JAX RPC possui os seguintes componentes que podem ser pontuados como constituintes de seu núcleo :

- disponibilizar API para definição de Web Services baseados em RPC, encapsulando complexidades inerentes ao SOAP e ao transporte da mensagem;
- disponibilizar API de ambiente de execução para a invocação de Web Services baseados em RPC que pudessem ser acessados nos seguintes modos :
  - stub (cliente) e ties (servidor);
  - proxy dinâmico;
  - invocação dinâmica de interface;
- empregar WSDL em JAX RPC para permitir a interoperabilidade com qualquer Web Services baseado em SOAP 1.1;
- disponibilizar mecanismos de mapeamento para tipos de dados entre Java e XML;
- suportar padrão para protocolos de transporte da Internet como HTTP;
- disponibilizar meios para serviços em pontos de acesso de servidor e serviços de clientes portáveis em vários aspectos da implementação JAX RPC [10].

Como premissa básica de disponibilização e publicação, Web Services baseados em JAX RPC devem ser acessados pelos clientes como servlets 2.2 ou em provedores de servidores J2EE 1.3 compatíveis [10] isto é publicados como componentes nos respectivos containers. Temos assim a possibilidade de publicar Web Services e clientes baseados na API JAX RPC como componentes J2EE [10].

No cenário de Web Services o emprego de JAX RPC é indicado para a parte de solicitação de serviço e de resposta de serviço, promovendo comunicação e executando solicitações e respostas baseadas em SOAP, trocando parâmetros, recebendo argumentos e retornando resultados em XML [10].

Sob a perspectiva do provedor de serviço, JAX RPC possibilita os mecanismos adequados para a exposição de componentes de negócios como Web Services [10]. Na outra extremidade cliente e consumidor de Web Services, os solicitantes de serviço, JAX RPC define meios e

formas para a invocação e o acesso a Web Services implementados em JAX RPC ou em qualquer outra tecnologia compatível com SOAP 1.1 e baseado em RPC para Web Services [10].

O funcionamento da JAX RPC inclui APIs para mapeamento entre tipos de dados realizando as conversões necessárias entre Java para XML e XML para Java, isto ocorre de forma concorrente e simultânea durante a execução dos processos de solicitação e resposta de Web Services (em pleno vôo), fazendo com que o mapeamento seja executado, como um terceiro processo, via utilização de threads da maquina virtual Java, para auxiliar o processo de comunicação.

Durante o processo de comunicação, ao invocar o Web Services JAX RPC, os parâmetros XML do cliente são automaticamente mapeados para objetos Java [10]. De forma similar, quando a resposta do servidor do Web Services é retornada, os objetos Java são mapeados para XML e devolvidos ao solicitante [10].

Mantendo a flexibilidade, JAX RPC possibilita ainda o desenvolvimento e a utilização, via mecanismo de encaixe (pluggable) de serializadores e desserializadores para suportar tipos e formas particulares, personalizadas de mapeamento entre Java para XML e XML para Java [10], estes módulos serializadores podem ser empacotados como serviços do próprio JAX RPC [10]. Adicionalmente, JAX RPC define esquemas e procedimentos de mapeamento entre WSDL e Java, através dos quais o documento WSDL disponibilizado e cedido pelo Web Services pode ser mapeado automaticamente para classes Java que serão empregadas no cliente como stub e no lado servidor como ties (amarras) [10].

4.3.2 – Modelo de aplicação e arquitetura JAX RPC

O modelo de aplicação e arquitetura do JAX RPC é composto de cinco elementos, agrupados de modo simplista, em dois serviços, um conceito, ferramenta de apoio ao desenvolvimento e implementação e ambiente de execução, a seguir passamos a descrever, pontualmente o funcionamento de cada uma destas partes.

- serviço de Web Services JAX RPC – representa o componente de negócio da aplicação que pode já encontrar-se implementado em Java, ser escrito em Java para atender ao Web Services ou produzido, a partir de uma ferramenta mais o WSDL de um Web Services qualquer, em classes Java[10] que deverão ser completadas funcionalmente, nesta opção teremos apenas o esqueleto das classes geradas. No ambiente J2EE o Web Services poderá ser implementado, publicado e

disponibilizado para uso como servlet, enterprise Java bean sem estado ou message-drive bean [10]. Durante a publicação, a disponibilização do Web Services, o serviço JAX RPC é assinalado com um ou mais pontos de acesso e configurado ligando-o ao protocolo de transporte. No presente, o serviço JAX RPC pode ser ligado ao protocolo HTTP e todas as mensagens são trocadas baseadas no funcionamento HTTP padrão de solicitação/resposta empregando-se os pontos de acessos configurados [10]. A definição de JAX RPC não dita e nem obriga que Web Services implementados empregando-se JAX RPC devam ser acessados e utilizados somente por clientes implementados em Java empregando esta mesma API, ao contrário, o desejável é que clientes sendo executados em linguagens diferentes de Java, plataformas e sistemas operacionais diversos, utilizem, sem problemas, Web Services implementados e disponibilizados com o emprego de JAX RPC [10].

- serviço consumidor de Web Services JAX RPC – representa o serviço cliente baseado em JAX RPC que deve acessar, utilizar e empregar na sua aplicação diária o Web Services disponibilizado [10] na rede que pode ser a Internet, a intranet ou a rede local. O serviço cliente consumidor de Web Services, neste tópico considerado em Java, deve ser percebido sempre de modo independente do alvo, da linguagem, da tecnologia em que foi implementado o provedor do serviço que disponibiliza o Web Services [10]. Nosso objetivo com este serviço cliente em JAX RPC é acessar, empregar e utilizar os Web Services disponíveis na Internet através de nossa linguagem preferida e amada Java. Não nos importando o que e como ocorram os detalhes, por trás do Web Services que desejamos utilizar e trazer para dentro do mundo Java. Isto significa que o Web Services que desejamos acessar, que pode ser um serviço implementado, funcionando e usando a plataforma Java ou qualquer outro baseado e compatível com SOAP 1.1, funcionando em plataforma não Java [10]. Para possibilitar a adaptação a este tipo de cenário, no lado cliente, JAX RPC define uma variedade de modelos clientes para invocação e acesso a Web Services, os quais empregam diferentes mecanismos como : modelo baseado em stubs, proxyies dinâmicos e invocação dinâmica de interface[10]. Clientes baseados em JAX RPC podem importar WSDL, expostos pelo servidor de Web Services, e produzir, automaticamente, código de classes baseadas em Java para acessar e utilizar o serviço [10].

- serialização e desserialização – durante o processo de comunicação entre solictação/reposta que ocorre no uso de um Web Services, JAX RPC emprega mecanismos serializadores e desserializadores para facilitar o mapeamento entre Java para XML e XML para Java, o que permite a conversão de tipos primitivos e objetos Java para representações baseadas em XML e vice versa [10]. Também é possível a criação de serializadores e descerializadores para tipos de dados personalizados, direcionados a aplicações e a resolução de problemas específicos. O que segundo [12] constituísse um prática que pode tornar o Web Services extremamente direcionado a um

conjunto restrito de clientes potenciais, retirando-lhe o seu principal caráter, que seria a universalidade e a facilidade de uso.

- ferramenta de apoio ao desenvolvimento e implementação (xrpcc ou wscompile) – com o objetivo de facilitar o rápido alcance por parte da tecnologia Java, reduzir a curva de aprendizado e criar aderência aos conceitos já estabelecidos, a API JAX RPC disponibiliza ferramenta (o nome da mesma é xrpcc mas deve mudar para wscompile nas próximas versões) com objetivo funcional de possibilitar a produção automática de arquivos de classes baseados em Java, necessários ao lado cliente e ao lado servidor, bem como o respectivo WSDL, de modo a promover a comunicação entre o servidor de Web Services e o cliente consumidor [10]. Objetivamente, a ferramenta produz o documento WSDL representando o Web Services a ser exposto, as classes Java em baixo nível de amarração do lado servidor (ties) e as classes stubs para o lado cliente [10], entretanto faz-se necessário que as classes e interfaces de implementação funcional, que realizam concretamente os Web Services a serem expostos, as classes que produzem o resultado concreto, sejam, efetivamente fornecidas para a ferramenta. Estas classes podem ser EJBs, servlets, Java Beans e outros componentes Java que serão encaixados com as classes produzidas pela ferramenta. No cenário cliente a ferramenta tem a possibilidade de produzir somente as classes Java de baixo nível stubs, proxy e invocação dinâmica de interface, a partir do WSDL exposto pelo fornecedor do Web Services que se deseja acessar [10] como já foi citado anteriormente.
- ambiente de execução JAX RPC – define o ambiente de execução e os mecanismos de funcionamento para os serviços de fornecimento de Web Services (lado servidor) e para o serviço cliente consumidor de Web Services, disponibilizando bibliotecas de execução necessárias e recursos do sistema [10]. Para a especificação atual JAX RPC recomenda containers servelet 2.3 ou maior ou ambientes compatíveis e baseados na especificação J2EE 3.1 no aspecto container servlet para a disponibilização de serviços de disponibilização de Web Services e serviços consumidores [10]. Em ambientes de servidores de aplicação baseados em J2EE os serviços de fornecimento e consumo de Web Services devem ser implementados como servlets em seus respectivos containers [10]. Entretanto, em versões recentes J2EE 1.4, especificações possibilitarão que a API JAX RPC venha implementar serviços utilizando EJB com sessão (sem estado) e message-drive bean. No que diz respeito ao protocolo de transporte o emprego do HTTP será estendido para permitir que JAX RPC utilize-o para gerenciamento de sessão e autenticação [10].

O modelo de aplicação e arquitetura JAX RPC poderá ser melhor visualizado na Figura 4.3.2-1 arquitetura de aplicação de Web Services baseado em JAX RPC. Na qual destacamos o acesso de dentro do ambiente J2SE, padrão usual e comum Java, a um Web Service implementado utilizando-se o modelo disponibilizado por JAX RPC.

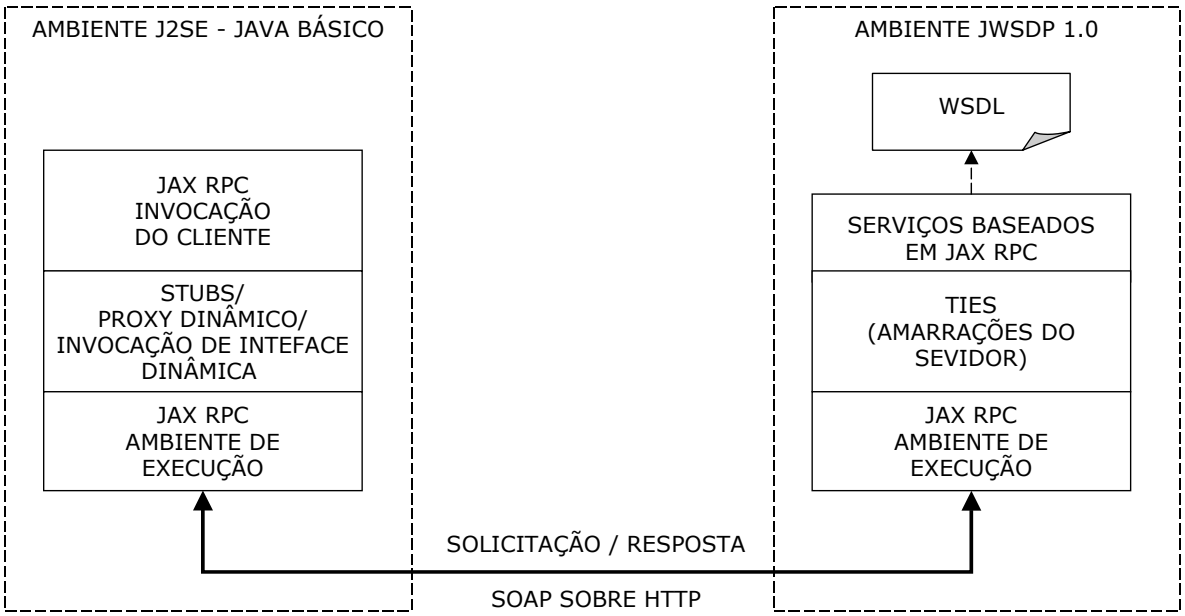


Fig. 4.3.2-1 arquitetura de aplicação de Web Services baseado em JAX RPC.

A estabilidade e aderência do API JAX RPC a tecnologia de rede da Internet, através de padrões Java estabelecidos, provém de seu projeto, que por concepção baseia-se no modelo de rede consagrado OSI (Open System Interconection) [11] . Temos então as seguintes características do modelo OSI implementadas concretamente em JAX RPC que passamos a pontuar, no sentido do meio físico para cima em direção a camada de aplicação :

- a camada física conduz, coloca concretamente a cadeia de bits através da rede;
- a camada de ligação de dados (data link) codifica e decodifica os pacotes de dados em bits para a camada física e toma os bits e transforma em pacotes para a camada de rede;
- a camada de rede é responsável por tarefas de chaveamento, roteamento, seqüência de empacotamento, endereçamento (destinação), prosseguimento e participação em circuitos virtuais, transmissão de dados de nó para nó;
- a camada de transporte providência a transferência transparente de dados entre hosts (computadores que se comunicam), responsabiliza-se pelo controle e recuperação de erros entre os dois pontos e controla o fluxo de dados na comunicação de modo que todos os nós possam ter suas capacidades e recepção/transmissão respeitadas. Obviamente, no ambiente de Web Services que estamos atualmente focando, as ligações HTTP com SOAP demonstram algumas fraquezas e limitações em alguns dos aspectos citados anteriormente e que provavelmente outras combinações e ligações não apresentarão como POP (Post Office Protocol), SMTP (Simple Message Transport Protocol), IMAP (Internet Message Acess Protocol) e JMS (Java Message Service);

- a camada de sessão estabelece, coordena e encerra conexões, trocas e diálogos entre as aplicações;
- a camada de apresentação, também denominada como camada de sintaxe, provê independência entre diferentes formatos de apresentação para os dados transformando-os da aplicação para a rede e da rede para a aplicação. Um das principais funções da camada de apresentação é transformar os dados provenientes da cada de rede em formatos aceitos, esperados e compatíveis que a camada de aplicação aceite;
- a camada de aplicação constitui-se na aplicação final do cliente, no processo usuário onde a funcionalidade do negócio esta endereçada. É a parte onde o dado transmitido pelo Web Services, via mecanismos JAX RPC, vai ser realmente empregado e terá sentido e validade para uso [12] . De modo pragmático, é para a camada de aplicação que todas as demais camadas trabalham.

Como pode-se verificar pela descrição e analogia das camadas com o modelo OSI, pontuado anteriormente, a API JAX RPC constitui-se em um protocolo altamente complexo, confiável e seguro que tem como característica encapsular toda a complexidade separando-a do desenvolvedor [12].

Sob a perspectiva do servidor, do provedor do Web Services, o desenvolvedor especifica o procedimento de acesso remoto através da definição de métodos empregando a interface de definição de serviços Java (procedimento já estabelecido com a tecnologia J2EE) e escreve, também em Java, uma ou mais classes Java que implementem os métodos planejados para ser expostos [12]. Então a API JAX RPC expõe estes objetos como pontos de acesso de serviço e produz, automaticamente, as amarrações (ties) necessárias [12] para as ativações destes como Web Services nos respectivos servidores, que podem ser containers servlets ou servidores J2EE.

O cliente nunca entra em contato ou mantém comunicação direta com a implementação do serviço de acesso ao Web Services [12]. Empregando stubs ou outro mecanismo para se comunicar com o ponto de acesso ao Web Services, invoca o serviço passando os parâmetros relevantes e recebe como retorno o resultado esperado [12]. Este comportamento e facilidade de uso são possíveis devido ao grau de abstração que a API JAX RPC possibilita. A Figura 4.3.2-2 modelo de camadas JAX RPC que apresentamos a seguir, relaciona a similaridade de projeto e implementação entre esta e API o modelo OSI. Em relação a ilustração anterior, temos a salientar que a presente enfatiza sobretudo detalhes do aspecto de rede abstraídos propositadamente na ilustração anterior.

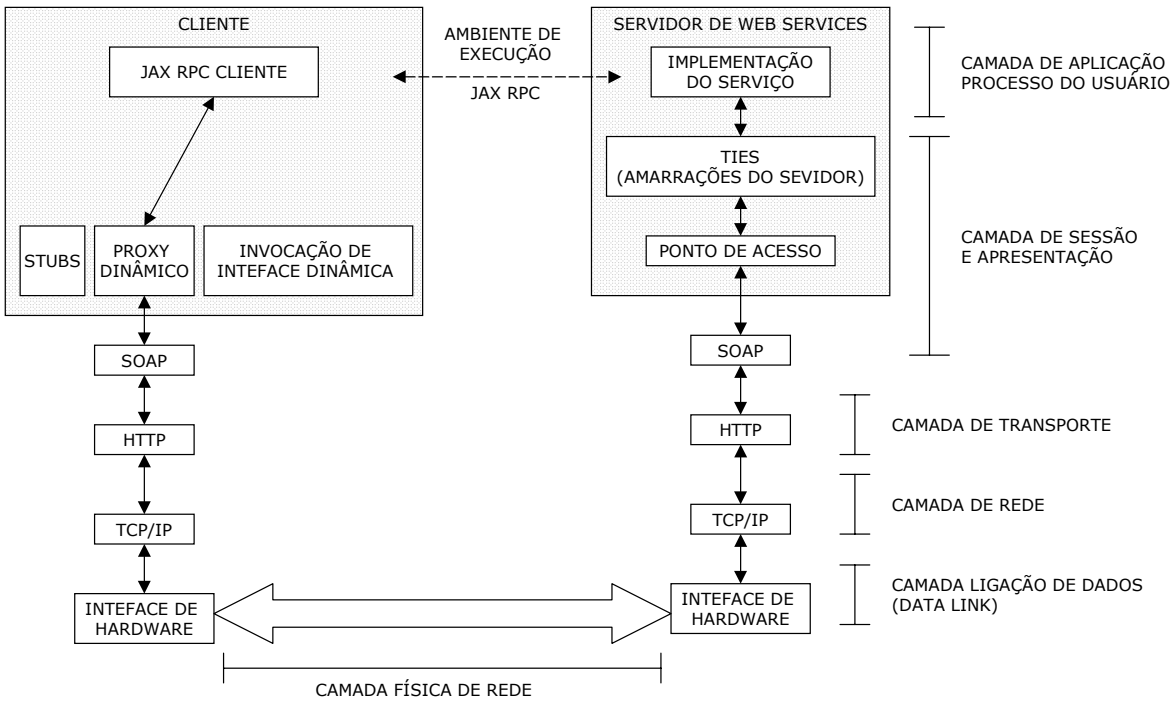


Fig. 4.3.2-2 modelo de camadas JAX RPC.

4.3.3 – Modelo de implementação baseado em JAX RPC

A especificação da API JAX RPC define ambos os lados para implementação de Web Services : servidor e cliente. No lado servidor, a exposição de Web Services dar-se a partir da disponibilização de componentes construídos em Java que ativam serviços dentro do ambiente de funcionamento definido.

O lado cliente refere-se ao modelo de implementação de clientes consumidores de Web Services que também são considerados serviços (de consumo), funcionando sob o ambiente especificado por JAX RPC. A definição do modelo de implementação de serviço do JAX RPC é orientada para a implementação baseada em mensagem RPC empregando ambiente de funcionamento ou ativando diretamente o serviço SOAP [10], dependendo da configuração adotada para utilização do JAX RPC .

O núcleo da API JAX RPC está concentrado no pacote Javax.xml.rpc que contém o pacote Javax.xml.rpc.handler.* adequado a implementação do ambiente de funcionamento [10] onde o mesmo é usado. E entre os serviços que funcionam neste ambiente podemos citar o pacote SOAP message handler API que é responsável em capturar mensagens SOAP do protocolo de transporte e encaminha-las para as camadas mais acima.

4.3.4 - Modelo de implementação do servidor de Web Services baseado em JAX RPC

Não existe na especificação da API JAX RPC nenhuma API para a implementação de serviços a partir desta [10]. Entretanto pode-se implementar Web Services baseados nesta API empregando-se duas maneiras distintas : a partir de classes Java (similar ao modo como se escreve aplicações RMI) ou através do uso de documento WSDL [10]. Em ambos os casos, a especificação JAX RPC não informa nenhum requisito necessário aos serviços de clientes consumidores para acesso aos serviços disponibilizados [10], mantendo o princípio básico da universalidade e interoperabilidade necessárias aos Web Services.

O desenvolvimento de Web Services a partir de classes Java com emprego de JAX RPC, torna-se bastante similar ao desenvolvimento de aplicações RMI [10], a seguir apresentamos de forma resumida os passos necessários :

- defini-se a interface remota do Web Services que exporá as classes e seus métodos para serem usados (definição do serviço);
- implementa-se as classes da interface remota para produzir concretamente o Web Services expostos (implementação do serviço);
- configura-se o serviço, neste passo informa-se em arquivo XML como o serviço será exposto, este arquivo informa a ferramenta de apoio como o Web Services deve ser produzido a partir das classes e interfaces fornecidas como base. A montagem deste arquivo constitui-se um dos passos mais importantes do processo;
- emprega-se a ferramenta que acompanha o kit de desenvolvimento de Web Services em Java para que a mesma produza as classes de amarrações (ties) para o servidor do Web Services e as classes stubs que servirão aos clientes Java para acesso ao Web Services. A ferramenta gera automaticamente o WSDL para o Web Services, porém tudo que a mesma deve realizar foi descrito no arquivo XML do ponto anterior;
- empacota-se e publica-se, expõem-se o Web Services produzido [10].

A segunda forma de implementar-se Web Services baseados na API JAX RPC é a partir de documento WSDL previamente exposto por Web Services ou produzido de alguma outra forma [10]. Neste caso, a ferramenta de apoio que acompanha o kit para o desenvolvimento de Web Services baseado em JAX RPC, importa o documento WSDL e produz as classes de serviço em Java necessárias

para a implementação [10]. Os passos chaves envolvidos neste método de implementação são os seguintes :

- configura-se o Web Services no arquivo XML que informa a ferramenta de apoio que as classes a serem produzidas o serão a partir do arquivo WSDL fornecido como entrada para a ferramenta. Todo esse arranjo deve ser informado neste arquivo de configuração. A montagem deste arquivo constitui-se um dos passos mais importantes do processo;
- produz-se, com a ferramenta de apoio a implementação de Web Services e o arquivo de configuração do ponto anterior, as classes Java do lado cliente, os stubs, e as classes Java de amarração (ties) do lado servidor;
- empacota-se e publica-se, expõem-se o Web Services produzido [10].

Nesse caso vale a ressalva de que as classes Java produzidas para o lado servidor são meras armações (templates) de classes Java, ocas, vazias de todo e qualquer código capaz de produzir e devolver os resultados esperados. Fica a cargo do desenvolvedor escrever o código que realmente faz com que as classes ou as armações de classes funcionem, fornecendo assim " vida" aos Web Services.

4.3.5 – Modelo de implementação do cliente de Web Services baseado em JAX RPC

A implementação de clientes Java para Web Services deve ser independente do alvo de implementação do serviço, segundo a especificação do JAX RPC [10]. O serviço cliente consumidor não pode depender do serviço provedor do Web Services ou se o mesmo está funcionando sobre plataforma e/ou ambiente Java e não Java.

Para o lado cliente, JAX RPC disponibiliza API específica e define modelo de acesso e invocação para o cliente Web Services [10]. O pacote disponibilizado para o lado cliente é o Javax.xml.rpc que constitui-se conjunto de interfaces e classes capazes de suportar as determinações JAX RPC pretendidas para que clientes invoquem Web Services dentro do modelo baseado em RPC e implementado pelo ambiente de funcionamento JAX RPC [10].

A partir das classes e interfaces do pacote Javax.xml.rpc pode-se criar clientes Java para Web Services com suporte para os três modelos de invocação definidos. A seguir apresentamos breve descrição pontual de cada um destes modelos.

- cliente baseado em stub – constitui-se no modelo de invocação de cliente Java para Web Services como modelo de programação mais simples entre todos. Este modelo emprega classes Java produzidas pela ferramenta de apoio do kit para desenvolvimento de Web Services. Criando instâncias destas classes, os clientes baseados em stubs acessam os Web Services definidos para estes stubs e interagem com as instâncias dos objetos provenientes do Web Services como se estivessem interagindo com qualquer outro objeto Java de forma transparente;

- cliente baseado em proxy dinâmico – estes clientes possibilitam a invocação do ponto de acesso, do alvo, do endereço de rede do Web Services dinamicamente, durante a execução do programa, sem a necessidade de classes do tipo stub. Este tipo de cliente emprega APIs de proxy dinâmico disponibilizadas pelo API Java Reflection. Particularmente o método getPort no pacote Javax.xml.rpc.Service interface permite que clientes baseados em proxy dinâmico sejam criados. Como citado em [12] este tipo de cliente destina-se a aplicações que utilizam Web Services que mudam continuamente de endereço, de ponto de acesso físico;

- cliente baseado em invocação dinâmica de interface – modelo de acesso a Web Services que permite ao cliente localizar dinamicamente o endereço físico do Web Services alvo, invocar e examinar os métodos disponibilizados pelo provedor de Web Services e todas essas tarefas em tempo de execução. Durante o tempo de execução em que procura dinamicamente Web Services que deseja "conhecer" o cliente emprega conjunto de operações e parâmetros, estabelecendo critérios para descobrir, achar, encontrar o serviço desejado e então invocar os seus métodos. Nesta procura clientes baseados em invocação dinâmica de interface são obrigados a realizar a invocação de serviços e seus métodos sem, entretanto, ter conhecimento prévio de seus tipos de dados, objetos e tipos de retorno [10]. Este é o método de acesso a Web Services mais complexo, com menor desempenho e mais raramente empregado em aplicações reais.

4.3.6 – Mapeamento Java / XML suportado por JAX RPC

Toda a complexidade de tipos existente no protocolo SOAP e em suas ligações com XML, são completamente abstraídas e encapsuladas em procedimentos padronizados pela API JAX RPC, através de facilidades de serialização e desserialização, que atuam automaticamente executando o mapeamento entre classes Java e dados XML e vice versa [10].

A especificação núcleo dos procedimentos de mapeamento entre tipos de dados Java e dados XML/WSDL, são descritos via XML Schemas 1.0, representados (XSD) e pela especificação de codificação SOAP 1.1 (SOAP-ENC) que podem ser encontradas na Internet e nas localizações

http://www.w3.org/2001/XMLSchema, http://www.w3.org/2001/XMLSchema-instance, http://schemas.xmlsoap.org/soap/encoding/ [10].

O processo de mapeamento é executado automaticamente pela ferramenta de apoio ao desenvolvimento de Web Services que a partir das classes de dados Java produz os devidos mapeamentos no WSDL, quando se implementa Web Services a partir de classes Java [10]. No caso em que os Web Services são produzidos a partir WSDL já prontos os dados são mapeados deste arquivo para as classes Java correspondentes [10].

No cenário de Web Services baseados em JAX RPC, quando um serviço é invocado, o ambiente de execução do JAX RPC transforma a chamada XML baseada em RPC para o seu correspondente representante em objeto Java e executa a solicitação do pedido sobre o objeto Java, a este processo nos referimos como desserialização (passagem de XML para Java, também chamado de Unmarshalling) [10]. Após a execução, o serviço retorna a chamada para o serviço cliente consumidor, transformando o retorno, o objeto Java em uma representação de dados baseada em XML, a este processo denominamos serialização (passagem de Java para XML, também denominado de Marshalling) [10].

A estrutura de suporte ao mapeamento entre Java/XML e XML/Java disponibilizada pela API JAX RPC procura atender a um amplo espectro de necessidades para a implementação de Web Services baseados em XML e RPC. Assim, o mapeamento de tipos considerados simples possui equivalentes atômicos para a conversão XML/WSDL. São considerados nativos mapeamentos de tipos de dados Java como int, long, float, double, short, boolean, byte, String, BigDecimal, BigInteger, Calendar e Date que encontram seus equivalentes definidos em XML/WSDL [10]. Tipos de dados que representam alguma característica de funcionamento agregada aos tipos considerados básicos, como por exemplo Arrays (vetores) também estão contemplados nos esquemas de conversão e pode-se proceder, de forma padrão e segura, o mapeamento entre Arrays Java e XML/WSDL e vice versa [10].

Para atender a tipos Java considerados complexos que possuem características elementares de estruturas, a API JAX RPC possibilita mapeamento para estruturas XML capazes de suportar tal complexidade, incluindo-se neste escopo suporte para componentes Java Beans constituídos de métodos getter e setter [10].

De acordo com a especificação SOAP 1.1, uma mensagem SOAP pode conter zero ou mais partes anexadas que estarão codificadas em MIME (Multipurpose Internet Mail Extensions). A possibilidade de transportar anexos foi uma das características do SOAP que o fizeram ser adotado

como o protocolo de mensagem padrão para Web Services. JAX RPC disponibiliza e permite o processamento de SOAP com partes anexas, processando este arranjo com o emprego do JavaBeans Activation Framework (JAF) [10]. O processamento do anexo do envelope SOAP ocorre durante a execução do JAX RPC que através do Javax.activation.Datahandler e Javax.activation.DataContentHandler, adquire acesso ao anexo através da classe DataHandler e pode manipular o conteúdo do anexo através do método getContent[10], que é empregado em outras situações que se faz necessário a manipulação de conteúdos considerados pouco convencionais como no caso o MIME.

4.3.7 – JAX RPC e J2EE

A evolução natural da tecnologia de Web Services é tornar-se parte integrante do ambiente de aplicação distribuída J2EE. Nas próximas versões aguarda-se a formalização de que a especificação do EJB (Enterprise Java Beans) 2.1 evolua atribuindo função para JAX RPC no ambiente J2EE de componentes e aplicações[10]. Isto quer significar que os demais servidores de aplicação compatíveis com J2EE irão implementar JAX RPC, o que permitirá a exposição de componentes J2EE como Web Services baseados em RPC [10].

Na versão EJB 2.1, temos a possibilidade de expor o EJB de sessão, sem estado, como Web Services empregando a Web Service Endpoint Interface a qual orienta-se pelas mesmas regras e requisitos já utilizados pela JAX RPC interface de serviço [10]. Esta semelhança quer significar que os métodos definidos na Web Service Endpoint Interface podem ser codificados na classe de implementação do bean. Outra modificação para que o EJB 2.1 possa ser exposto como Web Services foi a introdução, no descritor de publicação do bean, do elemento <service-endpoint> que contém o nome da classe que corresponde a interface do Web Services no ponto de acesso [10]. Para o tratamento de exceções geradas pela aplicação, está definido como responsabilidade do container mapear estas exceções para o mecanismo SOAP faults, conforme especificado em SOAP 1.1 [10].

A introdução de JAX RPC no ambiente J2EE permitirá que os componentes deste ambiente sejam acessados como Web Services a partir de clientes heterogêneos, incluindo clientes de aplicações Java e clientes de aplicações não Java [10]. Esta possibilidade representa para os Web Services fornecidos a partir do ambiente J2EE, vantagem qualitativa agregada, pois poderão dispor dos serviços já estabelecidos disponibilizados pelos containers como transações, segurança de aplicação, reusabilidade, para citar alguns [10].

4.3.8 – Interoperabilidade e JAX RPC

A questão da interoperabilidade, um dos objetivos maiores da tecnologia de Web Services, necessita de esclarecimentos e informações precisas, considerados fundamentais pela literatura, como citado em [10], [11] e [12]. O ponto central da interoperabilidade em Web Services repousa sobre a compatibilidade dos principais componentes da tecnologia : XML, protocolo de mensagem SOAP, protocolo de transporte HTTP, documento de descrição de Web Services WSDL.

No caso específico de Java, a API JAX RPC provedora de serviço, capaz de implementar e expor Web Services para consumo geral e heterogêneo pode interoperar (interagir, receber solicitação e responder) com qualquer serviço consumidor cliente que esteja empregando os padrões e as seguintes versões SOAP 1.1 e WSDL 1.1 compatíveis [10].

De modo similar, qualquer serviço consumidor cliente, produzido em Java pela API JAX RPC é capaz de consumir e interoperar (interagir, fazer uma solicitação e receber reposta) com qualquer Web Services disponível na Internet ou em qualquer outra rede de computadores que esteja utilizando para responder os padrões e as seguintes versões SOAP 1.1 e WSDL 1.1 compatíveis [10].

Esta, constitui-se uma, entre muitas, formas de definir-se a interoperabilidade de Web Services, com ênfase na solução proposta pela tecnologia Java com emprego da API JAX RPC.

Para garantir a interoperabilidade de JAX RPC com outras implementações providas por SOAP, torna-se importante a verificação da compatibilidade dessas com especificações e padrões como SOAP 1.1, WSDL 1.1, transporte HTTP 1.1 e XML Schema 1.1. Existem ainda outros requisitos e testes de interoperabilidade entre implementações SOAP disponibilizados pela Sun MicroSystems que podem ser experimentados em http://soapinterop.Java.sun.com [10].

4.4 – JAX M

A formalização de procedimentos referentes a protocolo para troca de mensagens baseados e definidos por SOAP 1.1 e especificações SOAP com anexos, a tecnologia Java disponibiliza a API JAX M [10]. Empregando infra-estrutura padrão disponível para troca de mensagem, JAX M utiliza conjunto de APIs em Java para facilitar o envio e o recebimento de mensagens assíncronas baseadas em XML dentro do ambiente de Web Services, fornecendo suporte adequado a esta

tecnologia, aos seus padrões e protocolos. Os principais objetivos relativos ao emprego da API JAX M no ambiente de Web Services são :

- possibilitar a implementação de aplicações XML capazes de manipular mensagens portáveis entre várias plataformas;
- disponibilizar mecanismos para comunicação e troca de mensagem síncrona (solicitação/resposta) e assíncrona (sentido único);
- transmitir e rotear mensagens entre muitos provedores;
- garantir a entrega de mensagens através de canais e mecanismos confiáveis;
- suportar padrões de protocolo da Internet como HTTP, SMTP e FTP [10].

Quando emprega-se JAX M sem o auxílio ou a presença da infra-estrutura de provedor, a mensagem pode ser enviada e recebida como no modo solicitação e resposta, através da conexão SOAP utilizando-se a API SAAJ (Soap with Attachment API for Java) 1.1, disponibilizada pelo JWSDP (Java Web Service Development Package) [10].

A noção de perfis de mensagens (message profiles) é disponibilizada pelo API JAX M, que permite a implementação de protocolos de mensagem sob SOAP. Uma aplicação cliente baseada am JAX M deve empregar perfil de mensagem para identificar as suas camadas inferiores de transporte e ligação, estrutura de mensagem e aspectos semanticos [10]. O perfil de mensagem deve ser empregado no ambiente de Web Services juntamente com JAX M para que a aplicação empacote, faça a embalagem de suas mensagens da maneira que considerar mais adequada. Desta forma temos o estabelecimento de acordo entre o cliente JAX M e o provedor de mensagens JAX M deste cliente [10].

Aplicações construídas no ambiente de Web Services baseadas em JAX M podem ser disponibilizadas e publicadas, tanto emissores quanto receptores de mensagens, como Java servlet 2.2 ou em containers compatíveis com servidores de aplicação J2EE 1.3. A API JAX M funciona como aplicação assíncrona e executa métodos ativados e estimulados por mensagens recebidas do provedor ou de um emissor, se for o caso [10]. Aplicações JAX M funcionando sem a infra-estrutura de provedores podem rodar também como clientes isolados e autônomos (standalone) trocando mensagens compostas de solicitações e respostas síncronas com outras aplicações compatíveis com JAX M ou SOAP 1.1 [10], o que mantém o princípio da heterogeneidade da tecnologia Web Services.

No cenário de Web Services, a infra-estrutura de mensagens JAX M comporta-se como intermediário no que diz respeito ao roteamento de mensagens em relação aos vários destinos [10]. O Web Services cliente, consumidor de serviço, solicita através de sua aplicação o envio de mensagens relativas ao Web Services desejado ao seu provedor de mensagens JAX M, o qual roteia a mensagem ao provedor de mensagens JAX M do provedor do Web Services solicitado. O provedor do Web Services solicitado, recebe a mensagem, executa o método apropriado conforme requerido pela aplicação solicitante ou poderá ainda rotear novamente a mensagem para outro intermediário ou para o destino final [10]. Neste caso, descrevemos a comunicação entre duas aplicações que utilizam a API JAX M com a infra-estrutura de provedor de mensagem JAX M disponível para cliente e para provedor do Web Services, esta é uma das várias situações que podem ocorrer. Entretanto, vale ressaltar que, dentro da tecnologia de Web Services, seria indiferente se qualquer dos lados estivesse utilizando ou não os benefícios e facilidades promovidos pela API JAX M, pois a comunicação se daria da mesma forma.

4.4.1 – Arquitetura JAX M

Conceitualmente, JAX M esta fortemente baseado no conceito de provedor [11], que é análogo ao MOM (message-oriented middleware, conforme citado em [10]). Considerando esta analogia, o provedor constitui-se em um simplificado sistema de corretagem, de recebimento e entrega de mensagens no meio de dois pontos que trocam estas mensagens [11]. O provedor fornece os serviços necessários para persistência e o encaminhamento a diante de mensagens com confiabilidade, possibilidade de roteamento e infra-estrutura adicional relativa para funcionalidade de serviços de qualidade como segurança, alta disponibilidade e escalabilidade [11]. No caso da arquitetura para JAX M o requisito mais importante, além das características de provedor descritas anteriormente, é que o mesmo suporte SOAP com anexos e HTTP [11].

As raízes do conceito de provedor JAX M, estão fincadas na experiência e amadurecimento de JMS (Java Message Service), pois JAX M posiciona-se no topo da implementação do provedor [11] que deve funcionar em harmonia e estreita aderência com a API JAX M, porém tornando-se para esta uma estrutura de apoio confiável.

O provedor desempenha o papel do proxy, no sentido de que a aplicação emprega o provedor fazendo com que todas as mensagens passem pelo mesmo até chegarem a ela [11]. Assim, quando uma aplicação baseada em JAX M envia uma mensagem, ela primeiramente é entregue ao provedor, que por sua vez a entrega ao seu destino ou aos seus destinos finais de acordo com a determinação da aplicação [11]. De modo similar, quanto a aplicação JAX M recebe uma mensagem,

esta anteriormente já foi recebida pelo provedor e encaminhada para a aplicação [11]. Um provedor, emprega tipicamente os serviços disponibilizados pelo container J2EE, considerando-se que o container possui como requisitos :

- interface para interagir com o mundo externo. Esta estrutura é possibilitada por servlets ou message-driven EJBs;
- mecanismos que possam ser administrados e controlados pelo ambiente de execução. Este requisito envolve vasto conjunto de objetos de apoio a infra-estrutura do ambiente, de possam ser administrados e controlados como JNDI, factories e outros (como objetos utilizados e administrados pelo JMS) [11].

Diante do exposto anteriormente, sob determinadas perspectivas, e em relação a alguns momentos de operação e utilização do provedor o mesmo pode ser classificado, funcionamente, como estando situado entre servidor de correio eletrônico e corretor de MOM, porém esta combinação de funções e desempenho constituem o que denominamos de provedor JAX M [11], para melhor facilitar o desenvolvimento deste tópico.

A caracterização do conceito de provedor JAX M poderá ser melhor visualizada na Figura 4.4.1-1 modelo conceitual para provedor JAX M apresentada a seguir, na qual observa-se a presença dos padrões que regem a interoperabilidade da tecnologia de Web Services no que se refere ao SOAP com anexos e ao protocolo de transporte HTTP, o ponto a frisar é que tanto os blocos de envio de mensagem e de recebimento, por se tratar de ambiente de Internet e tecnologia Web Services, podem estar escritos e funcionando em qualquer plataforma compatível com os padrões Web, conforme preconiza [10].

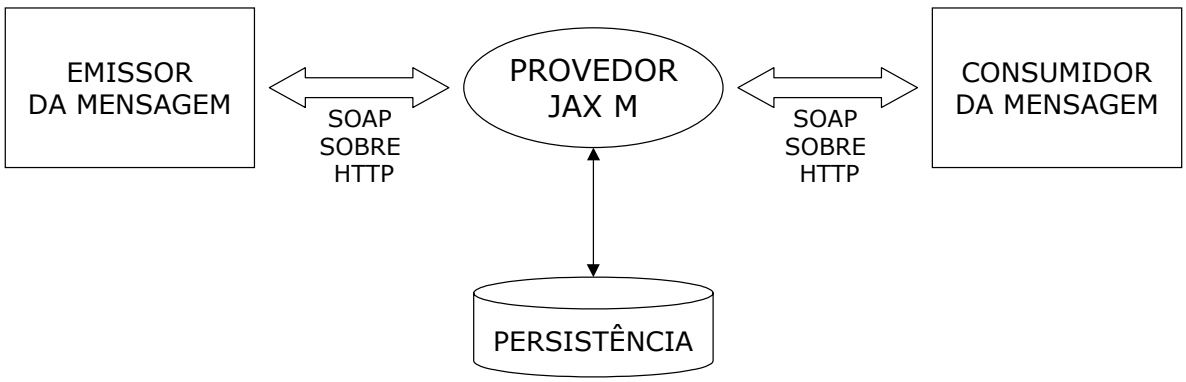


Fig. 4.4.1-1 modelo conceitual para provedor JAX M.

Adotando a metodologia de pilhas, preconizada em [12] e também em [11], para melhor demonstrar como aplicações baseadas na API JAX M mantém relacionamento funcional hierárquico com a infra-estrutura de provedor, oferecemos a Figura 4.4.1-2 pilha da arquitetura JAX M. Observamos nesta ilustração a presença do elemento provedor, próximo ao nível de comunicação, porém completamente controlado e mantido pela API JAX M. Outro elemento que merece destaque são os blocos de perfis que permitem a personalização e adaptação de aspectos da comunicação destas aplicações com a API JAX M.

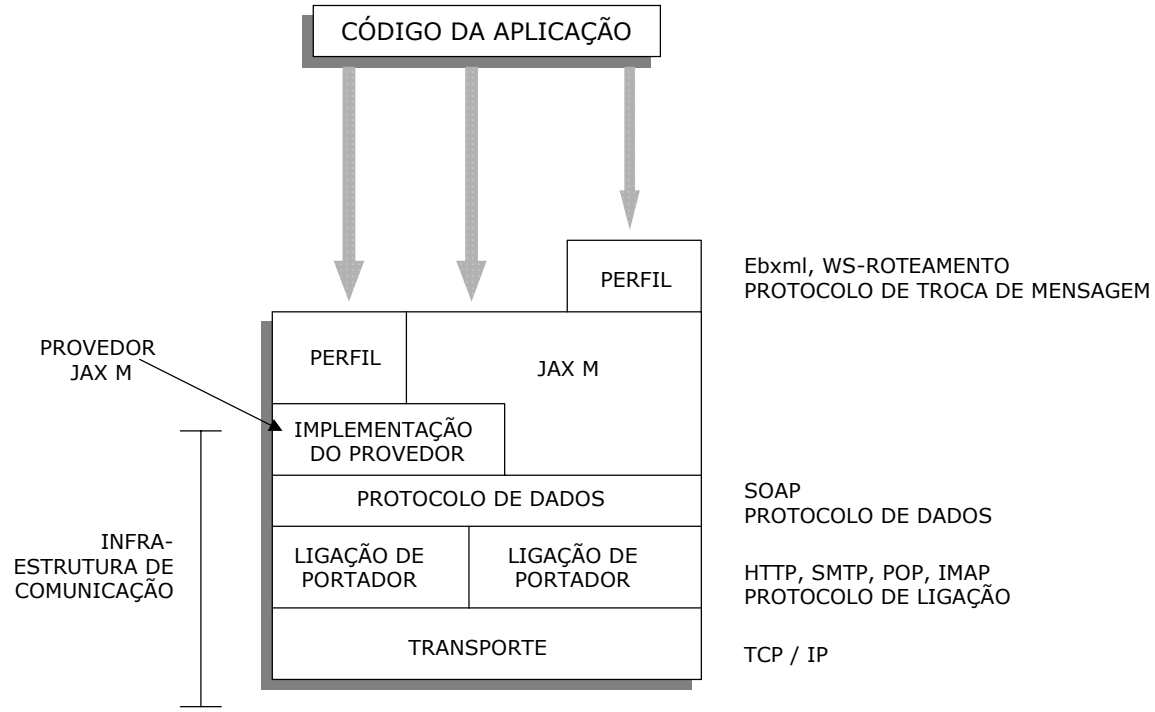


Fig. 4.4.1-2 pilha da arquitetura JAX M.

4.4.2 – Componentes da API JAX M

A principal tarefa da API JAX M constituiu-se no desenvolvimento e no estabelecimento de padrões de interfaces Java para aplicações que empregam a funcionalidade e a utilização de serviços na troca de mensagens baseadas em XML utilizando protocolo de mensagem SOAP [11]. Este padrão divide a API em duas partes :

- Javax.xml.soap – representa a implementação, o empacotamento e a especificação para SAAJ. Inicialmente está API fazia parte da especificação do próprio JAX M, mas por causa do uso da mesma por várias outras API, como JAX RPC, SAAJ foi movida para uma especificação independente. SAAJ foi desenvolvida para criar, manipular, enviar, receber mensagens SOAP [11] com ou sem anexos e com ou sem provedor JAX M;

- Javax.xml.messaging - define o núcleo de especificação da API JAX M e interações com o provedor de mensagens que estiver sob seu controle [11].

4.4.3 – Detalhamento de perfis (profiles da API JAX M)

Matendo a mesma filosofia de construção de JMS, a API JAX M disponibiliza somente um mecanismo bastante simplificado e de processamento muito leve para conexão e comunicação com o provedor, sem impor nenhum requisito ao formato ou conteúdo da mensagem, além de que a mesma esteja de acordo com o padrão SOAP [11].

Entretanto, nas aplicações do mundo real, faz-se sempre necessário e indispensável o rigoroso controle e o prévio acordo entre as partes sobre o tipo de informação e sobre qual a estrutura da informação será passada através da API JAX M com ou sem provedor [11].

Os padrões estabelecidos SOAP 1.1 e SOAP com anexos disponibilizam o modelo básico de pacote. O requisito para o emprego e utilização deste pacote resume-se na exigência de que toda e qualquer mensagem transmitida entre o ponto A e o ponto B seja composta e tenha como invólucro externo um envelope que contenha um header e um body [11]. Isto é básico e simples. Entretanto para o perfeito sucesso de comunicação entre partes empregando mensagens XML, as partes que se comunicam devem ser capazes de realizar as seguintes tarefas, descritas nos perfis através de protocolo XML comum associado a XML Schemas, as quais podem ou não compor os mesmos :

- endereçamento – partes devem concordar sobre o esquema padrão de endereçamento e disponibilizar mecanismos para correlacionar solicitação para e resposta de ou seja, possibilitar a automação do comportamento que correlaciona remetente e destinatário da mensagem;
- conteúdo processado – partes devem ter conhecimento do que a mensagem representa e significa no contexto de negócios aos quais estão vinculadas. Um documento deve estar relacionado a determinadas regras e todas as partes envolvidas no transporte deste documento devem ter conhecimento destas regras ou seja devem "saber" o que deve ser feito, qual o procedimento relacionado a cada documento e suas regras específicas;
- processamento de header – as partes devem estar aptas a examinar os atributos dos headers SOAP de mensagens que forem recebidas e processar, quando for o caso, alguns metaconteúdos ou metadados que indiquem interferência necessária na mensagem;

- checagem de segurança e criptográfia / descriptográfia – a autenticidade das mensagens deve ser verificada pelas partes, para que seja determinada se a mensagem recebida é de origem e emissor devidamente autenticados, que tenha sido autorizado a envia-la como tarefa e que os dados criptografados estejam seguros, invioláveis e confiáveis;
- manipulação de exceções e erros – os envolvidos na comunicação devem garantir que os emissores de mensagens recebam notificações apropriadas e consistentes sobre falhas e erros que possivelmente ocorram quando mensagens recebidas não contenham informações acuradas ou suficientes para serem processadas com normalidade;
- roteamento – partes devem ser capazes de rotear mensagens, isto significa possuir habilidade de manipular mensagens verificando suas partes lógicas para determinar qual o recipiente de destino da mesma, determinar muito rapidamente, qual a rota mais próxima até ele e encaminhar a mensagem para esta rota, de modo a contribuir para que ela seja entregue ao recipiente final de destino o mais rápido e pelo menor caminho possível [11].

Estes são alguns dos pontos que um perfil pode ou não contemplar, dependendo do grau de requisitos da aplicação no emprego da API JAX M. Observa-se nos pontos descritos anteriormente que a API JAX M não descreve a semântica que endereça os requisitos e nem os direcionamentos que governam o protocolo de troca de mensagens SOAP/XML [11]. Todos os pontos são rigorosamente descritos pelo protocolo comum e básico XML definido pelo que denominamos de perfil (profile) e pelo XML Schemas que está associado a este perfil (profile) [11].

Um perfil constitui-se realmente em um padrão confiável da indústria para a troca de mensagens, como ebXML Message Handling Service que pode ser definido por organizações como OASIS ou W3Cs XMLP [11]. A real diferença entre os perfis definidos pela indústria esta nas regras bem estabelecidas para processamento de mensagens e a representação em alto nível de abstração sobre o protocolo SOAP que governa e dirige a forma como parceiros de negócios poderão utilizar mensagens XML para operar seus relacionamentos [11].

Com a adição da camada de pefil (profile) no conjunto de componentes que operam junto ou encaixados (plugados) com a API JAX M, a aplicação que necessita do processamento especial proporcionado pelo perfil (profile) utiliza a API específica do perfil (profile) e não diretamente a API JAX M [11] desenvolvida para ser genérica e base referencial. Assim, uma aplicação que necessite de um perfil especial empregará a API especializada fornecida pelo fabricante deste perfil e não diretamente a API JAX M, neste caso é fácil perceber que a API JAX M passará a mensagem para a API do perfil específico que por sua vez a entregará para a aplicação, com na Figura 4.4.1-2.

Existem casos em que os desenvolvedores de perfil optam por não desenvolve-los sobre a API JAX M, nestes casos os perfis são fornecidos completamente e realizam também as funções da API padrão JAX M. Um destes casos é o perfil definido pelo ebXML Message Service Specification que encarrega-se de todo o processo colocando e retirando a mensagem SOAP na camada de rede [11].

Os perfis (profiles) possuem como característica fundamental de implementação forte aderência e ligação com os headers do procolo SOAP, isto ocorre porque os headers contém a maioria das informações específicas para os perfis (profiles). A carga da mensagem ou o conteúdo de negócio armazenado no elemento body da mensagem SOAP, geralmente permanece inalterado, não importando o protocolo empregado para o transporte e a troca de mesma[11]. Em resumo, o perfil (profile) e todos os esquemas associados ao mesmo, descrevem o conjunto de regras e procedimentos que todas as partes envolvidas na comunicação devem entender e concordar para que a comunicação, via mensagem XML, torne-se efetiva [11].

4.4.4 – Projeto de aplicações com JAX M

No que se refere as opções de arquitetura a serem adotadas, aplicações orientadas ao processamento de mensagens baseadas em JAX M podem ser classificadas em dois grupos disjuntos que determinam suas capacidades no manuseio de mensagens, são eles : aplicações que não empregam provedor JAX M, aquelas que operam com mensagens síncronas. E aplicações que utilizam e contam com a efetiva estrutura do provedor JAX M, aquelas que operam mensagens assíncronas [11]. Cada uma dos grupos possui emprego e utilização em cenários que adaptam-se as suas características naturais.

As aplicações que utilizam o esquema de troca de mensagens síncronas, sem que haja a presença e o suporte do componente provedor, tem como requisito fundamental a disponibilidade, o engajamento simultâneo de todos os participantes da comunicação, no momento em que se dará a troca de mensagens[11]. A esta modalidade de comunicação denomina-se solicitação-resposta síncrona (ou comunicação ponto a ponto). O cliente envia a mensagem e inicia um processo de espera, de congelamento, fica completamente parado, aguardando a resposta. Isto ocorre até que o tempo determinado para a espera se esgote, o que gera um alerta de falha ou que seja recebido uma resposta que pode ser relativa ao conteúdo de comunicação, mas também pode ser um comunicado de falha. O cliente não pode executar nenhum outro trabalho ou tarefa enquanto aguarda a resposta [11]. A Figura 4.4.4-3 comunicação síncrona de mensagem ponto a ponto de um cliente sem provedor, oferecida a

seguir, tem como objetivo ilustrar a situação que ocorre genericamente ao adotar-se este modelo para a troca de mensagens. Observa-se que a API JAX M está presente em ambos os lados, porém no lado da organização B, comporta-se, conforme proposto em seu projeto original, de forma pouco acoplada e leve com relação a estrutura de provedor da qual desfruta os serviços.

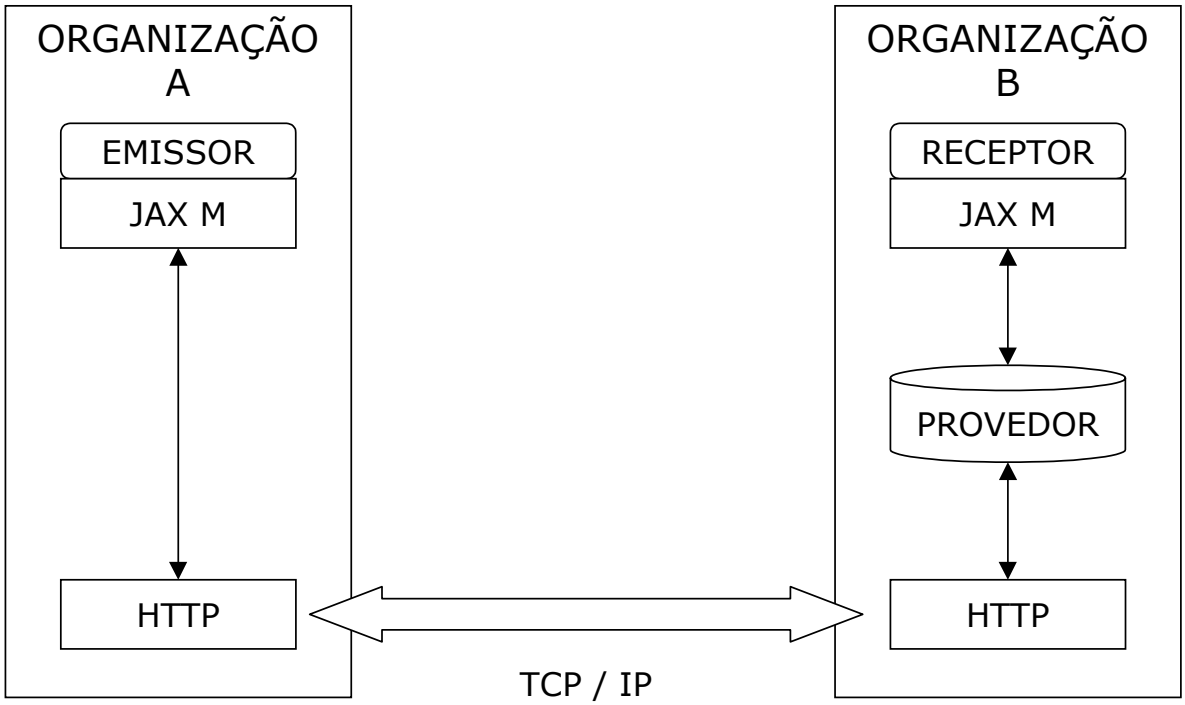


Fig. 4.4.4-3 comunicação síncrona de mensagem ponto a ponto de um cliente sem provedor.

Em nível mais específico e concreto, relativo ao contexto de Web Services e da comunicação real entre computadores, a comunicação síncrona entre emissor e receptor pode ocorrer em duas categorias distintas :

- síncrona com resposta – o emissor é bloqueado, fica aguardando, sem realizar nenhuma outra tarefa, até que a resposta para a mensagem enviada seja recebida. Esta modalidade de comunicação é geralmente implementada em casos de uso onde ocorrem mensagens somente de leitura, no padrão denominado síncrono-inquisitivo (synchronous-inquiry), no qual os dados no lado receptor não são trocados e nem alterados de nenhum modo como resultado da solicitação, invocação [11]. Essa categoria de comunicação é empregada quando o lado emissor solicita documentos do lado receptor, que são devolvidos, como cópias, sem que ocorra alterações no lado que recebeu a solicitação;
- síncrona com aviso de recebimento – nesta modalidade o receptor não envia de volta uma resposta completa, apenas envia sinal indicando que recebeu, que a mensagem foi aceita e que se encontra no receptor. O Cliente permanece congelado, parado, esperando pela confirmação de recebimento

do receptor e enquanto isso, não executa nenhuma outra tarefa. Entretanto retornos podem ser recebidos em ordem diferente das mensagens emitidas pelo emissor, porém geralmente obedecem a seqüência de pares mensagem/confirmação [11]. O propósito desta modalidade de comunicação é primeiramente, notificar o cliente de que a mensagem foi recebida pelo destinatário e em segundo desbloquear o emissor liberando o thread que fica no aguardo da resposta consumindo recurso computacional. Este arranjo é sempre implementado em casos de uso correspondentes ao padrão denominado síncrono-alteração (synchronous-update). Neste padrão temos a possibilidade de promover trocas e alterações nos dados de negócio armazenados no destinatário da solicitação, que podem ser, potencialmente compartilhados, e usados por outros serviços e consumidores em outras instâncias de recebimento dos mesmos dados [11]. Nesta categoria de comunicação pode-se alterar uma informação em um dos clientes do serviço e demais terão acesso a mesma assim que ela for confirmada para quem solicitou a alteração e tornar-se disponível.

Para o contexto de Web Services a comunicação síncrona com resposta encontra várias aplicações, principalmente no que se refere a troca de documentos entre as partes. Esta modalidade torna-se útil e caracteriza-se pela solicitação que pode ser uma mensagem curta sobre a necessidade de receber um documento extenso como um contrato ou pode ser um documento extenso que precise de uma resposta curta como um aprovo ou alguma outra alteração. O ponto a ressaltar é que em ambos os casos temos o envio de informação semanticamente importante relativa a natureza do contexto de negócio. A Figura 4.4.4-4 mensagem síncrona com resposta, permite a visualização da troca de informações entre os participantes, no caso temos, mesmo que de tamanhos e formas diferenciadas a troca de documentos entre as partes que devem ser reconhecíveis por ambos no contexto de negócio.

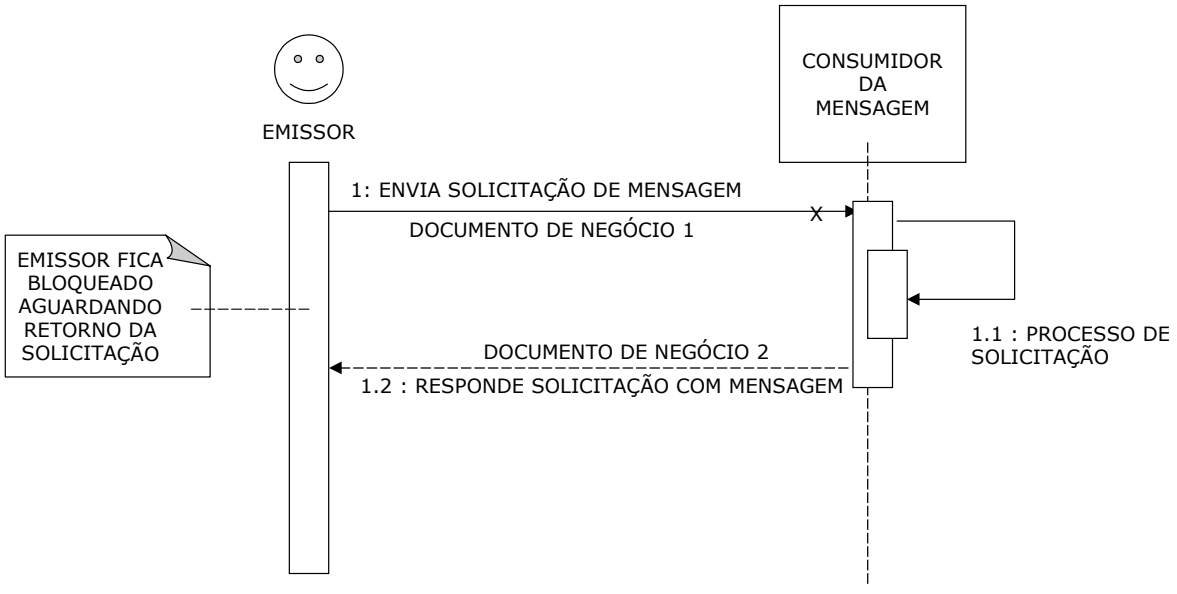


Fig. 4.4.4-4 mensagem síncrona com resposta.

Existem muitas aplicações para a tecnologia de Web Services nas quais parceiros de negócios comunicam-se de forma dinâmica empregando padrões já estabelecidos e que exigem agilidade e confiabilidade. Nestes contextos a categoria de comunicação síncrona com aviso de recebimento torna-se a solução canônica mais adequada, pois com o emprego da mesma, o emissor envia para o destino mensagem previamente acordada e aguarda a confirmação de recebimento que deve ser retornada com máximo de rapidez possível, a fim de agregar escalabilidade e agilidade ao processo que já, por natureza de concepção, possui a característica de confiabilidade de entrega e recebimento. Nesta modalidade de comunicação, espera-se que o receptor responda rapidamente o recebimento da mensagem, confirmando para o emissor que a mesma foi aceita e dispensando o emissor de ficar no aguardo desta confirmação no menor tempo possível. Feito isso, então o receptor da mensagem iniciará os procedimentos internos para o processamento adequado da mensagem recebida.

Na modalidade de comunicação de mensagem síncrona com confirmação de recebimento a característica principal do padrão no fluxo de mensagem é a de que a mensagem emitida trata-se de documento reconhecido no contexto de negócio que deve ser processado de alguma forma pelo receptor (pelo menos armazenado). Entretanto o resultado, a confirmação de recebimento do documento anteriormente recebido pelo receptor deve, necessariamente, constituir-se de mensagem curta, objetiva, identificando o documento recebido, com características de comando, para informar ao emissor do documento, que o documento enviado foi recebido e encontra-se no receptor (emissor da confirmação). Mesmo com a redundância necessária do português, oferecemos a Figura 4.4.4-5 mensagem síncrona com confirmação de recebimento na qual enfatizamos a diferença entre a mensagem de negócio enviada e a mensagem de confirmação de recebimento retornada.

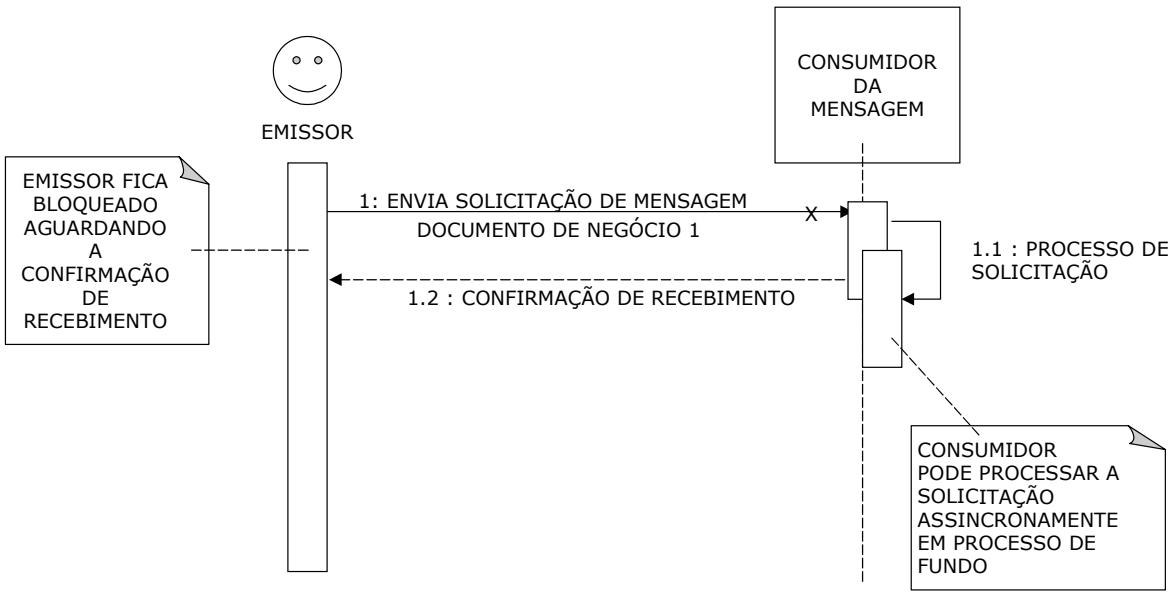


Fig. 4.4.4-5 mensagem síncrona com confirmação de recebimento.

Uma aplicação que utiliza como estrutura de base provedor JAX M pode disponibilizar suporte para ambos os tipos de aplicações orientadas a mensagens : aplicações orientadas a mensagens síncronas e aplicações orientadas a mensagens assíncronas, isto ocorre devido ao comportamento do provedor JAX M que funciona como um MOM ou corretor de mensagens [11].

No modo assíncrono para comunicação de mensagem, o cliente envia a mensagem e continua processando e executando suas tarefas normais sem entretanto perder tempo em aguardar qualquer tipo de resposta [11]. Baseado em algum forma de acordo de prioridade, definido previamente ou através de negociações eventuais, o serviço de transporte de mensagem, o provedor JAX M, não está submetido a qualquer requisito ou restrição para janela de tempo na qual a entrega deva ser feita ou qualquer outro tipo de restrição no que diz respeito aos aspectos físicos como a obrigatoriedade de utilizar-se o mesmo transporte ou conexão [11]. O provedor deve apenas receber a mensagem de forma confiável e garantir a sua entrega ao destinatário correto.

Este comportamento por parte do provedor JAX M, faz com que apareça o problema da correlação ou seja como relacionar a resposta a solicitação prévia, como saber de quem é solicitação feita a alguns dias dona de uma resposta entregue ao provedor JAX M hoje [11]. Este problema, não faz parte do universo de comunicação síncrona, pois ambos os interessados em trocar mensagens estão conectados e identificados on-line, no momento da troca de mensagens. A API JAX M não especifica absolutamente nada sobre este problema, não indica nenhum modo de como a correlação pode ser construída, deixando esta solução para ser implementada na camada de perfil (profile) que deverá oferecer a solução para este impasse [11].

Para que ocorra o modo assíncrono de comunicação faz-se necessário e suficiente que ambos participantes contem com a estrutura de apoio do provedor, conforme pode-se observar na Figura 4.4.4-6 papel funcional do provedores na comunicação assíncrona de mensagens, as organizações possuem provedor incorporado em suas estruturas sob o gerenciamento e controle da API JAX M.

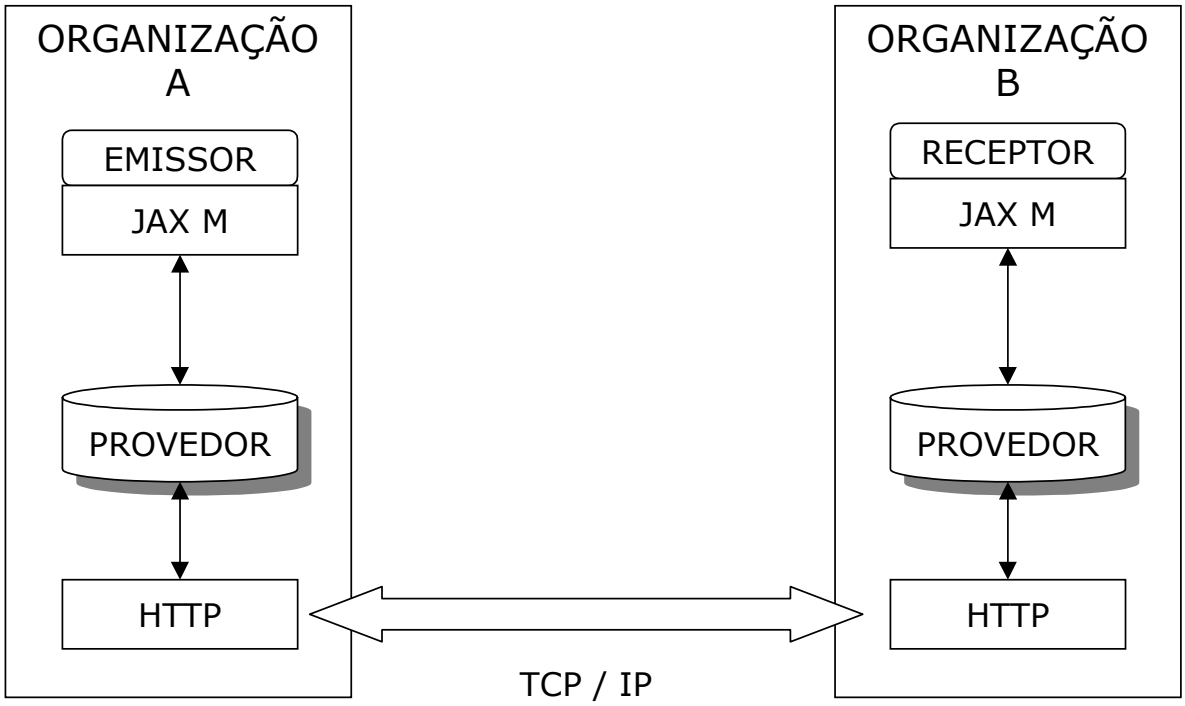


Fig. 4.4.4-6 papel funcional dos provedores na comunicação assíncrona de mensagens.

A comunicação assíncrona, demonstra-se acompanhada de mais sofisticação que a comunicação síncrona. Considerando seu uso e emprego no contexto operacional e prático da implementação de aplicações orientadas a mensagens, podemos classificar sua utilização em três categorias bem definidas :

- assíncrona com resposta – o cliente envia a resposta para o recipiente destino e aguarda uma resposta. Entretanto, o cliente aguarda a resposta mas não interrompe seu processamento para aguarda-la, continua sua execução normal [11], o processo que aguarda a resposta fica a cargo do provedor JAX M. O recipiente assíncrono, destino da mensagem enviada, envia de retorno o resultado para a mensagem ao emissor de origem, de modo assíncrono, quando completa o trabalho solicitado em seu lado, sem limitação de tempo para que o mesmo seja realizado. Esta categoria de comunicação assíncrona de troca de mensagens é sempre implementada em casos de uso correspondentes ao padrão assincrono-inquisitivo (asynchronous-inquiry) que caracteriza-se

pela operação sobre dados somente de leitura ou seja os dados trocados e recebidos de ambos os lados nunca são mudados;

- assíncrono com aviso de recebimento – similar ao modo assíncrono com resposta, neste modo a diferença consiste em que o consumidor da mensagem envia uma confirmação de resposta para o emissor da mensagem indicando que a mesma foi recebida e aceita, correlacionando-a com a mensagem recebida anteriormente. A semântica da correlação não é especificada em nenhum momento pela API JAX M e tipicamente fica sob a responsabilidade da cama perfil (profile). Nas situações onde ocorre a necessidade de comunicação assíncrona de mensagens, com a restricão de confirmação de recebimento, temos o caso de uso padrão denominado assíncrono-alteração (asynchronous-update) no qual dados de negócios, potencialmente compartilhados e consultados por vários outros usuários e consumidores podem sofrer trocas, mudanças e alterações no lado receptor da mensagem e portanto faz-se necessário o grau de confiabilidade disponibilizado por este padrão;

- mensagem em sentido único - O cliente envia a mensagem e não fica travado ou a espera de alguma forma de resposta por parte do destinatário. Após o envio o cliente emissor fica livre para continuar o seu trabalho. Este padrão de comportamento também pode ser observado como o padrão assíncrono-alteração (asynchronous-update) no qual dados no receptor podem ser alterados. Entretanto, devido ao detalhe de ausência de confirmação, esta categoria de comunicação assíncrona de mensagens possui aplicações limitadas, devido ao grau de confirmação e confiabilidade [11]. Esta categoria também é conhecida como atire e esqueça ! (fire and forget !), como citado em [10].

As categorias de comunicação no modo assíncrono contam com o aparato do provedor JAX M, o que as torna mais robustas e confiáveis, se comparadas ao modo de comunicação síncrono. Entretanto temos a observar que cada um destes modos possui, pelo menos duas característica em comum, o retardo entre o envio e a resposta. O tempo no qual o emissor envia a mensagem e o tempo em que ele recebe a resposta. E a necessidade da disponibilização de um processo ou mecanismo equivalente para que esteja sempre disponível a receber as respostas assíncronas que podem ser enviadas.

O processo responsável em estar sempre a disposição para receber as resposta assíncronas que podem ser recebidas a qualquer tempo fica sob responsabilidade da camada do provedor, controlado pela API JAX M. No que se refere a diferença de tempo entre o envio e o recebimento da resposta, principal característica da comunicação assíncrona de mensagens, estas podem ser visualizadas com perfeição. A Figura 4.4.4-7 comunicação assíncrona de mensagem com

resposta, demonstra que ao receber uma solicitação para o processamento de um documento de negócio no tempo T1 a resposta, assíncrona, somente é enviada em retorno no tempo T2, com T1 < T2. Esta modalidade de comunicação trafega sempre entre emissor e receptor, documentos de negócio com conhecimento e significado semântico em ambos os lados.

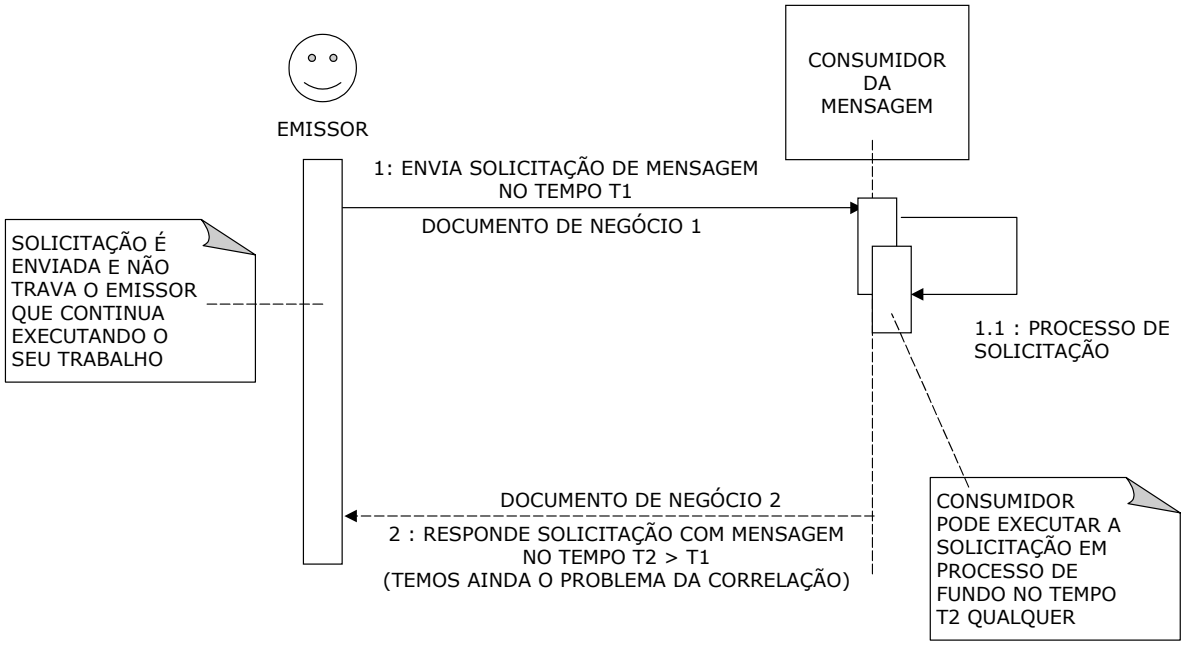


Fig. 4.4.4-7 comunicação assíncrona de mensagem com resposta.

Em categorias de comunicação de mensagens assíncronas, temos a diferença entre o tempo de recebimento da solicitação T1 ser sempre menor ou muito menor que T2, tempo em que ocorre a resposta. No caso da comunicação assíncrona de mensagens com confirmação de recebimento, encontramos ainda como característica do fluxo de mensagem o fato de a solicitação tratar-se de documento pertencente ao contexto de negócio, conhecido por ambos os lados e a confirmação, caracterizar-se por mensagem compacta relativa tão somente ao comunicado de recebimento do documento de solicitação. Isto deve ocorrer para que seja mantida a característica de confiabilidade na entrega da mensagem e a escalabilidade e agilidade, aqui uma questão um tanto delicada, que deve ser analisada sob alguns aspectos, já que o tempo para a emissão da confirmação de recebimento, pode ser um T2 qualquer, de qualquer tamanho. Temos ainda o problema da correlação entre mensagens. A Figura 4.4.4-8 comunicação assíncrona de mensagem com confirmação de recebimento, oferecida a seguir, demonstra como estas características estão relacionadas e enfatiza de modo discreto uma possível destinação que o emissor da mensagem de solicitação pode dar a confirmação de recebimento.

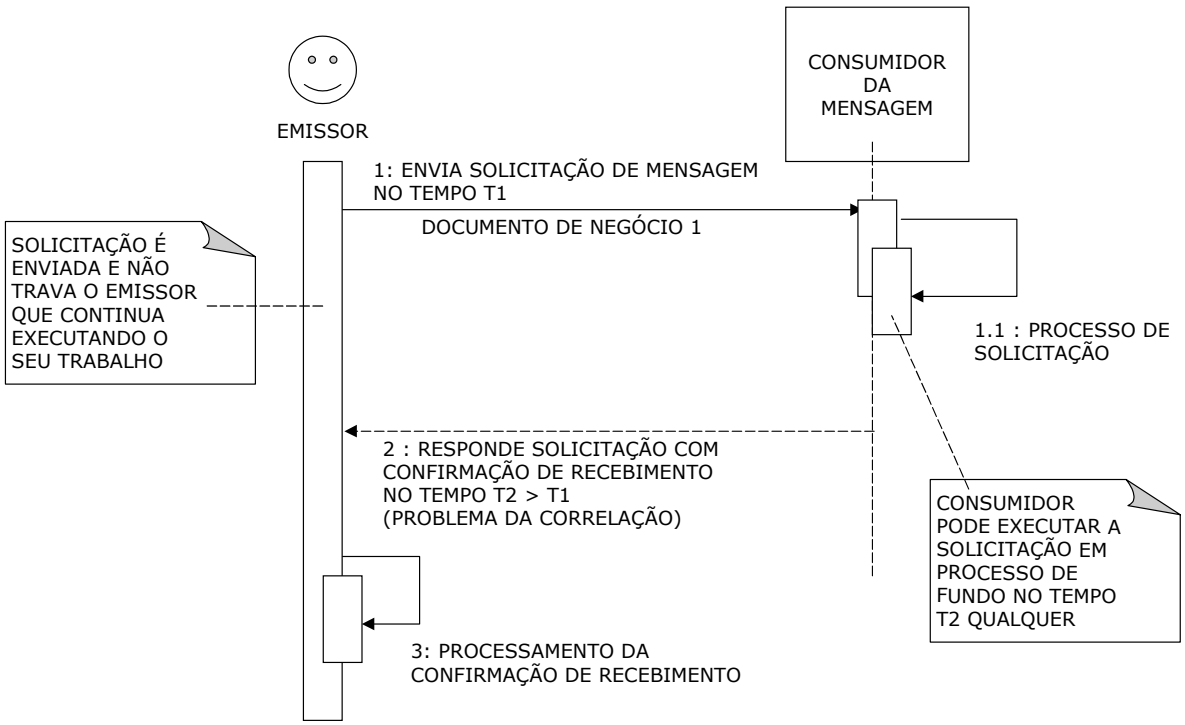


Fig. 4.4.4-8 comunicação assíncrona de mensagem com confirmação de recebimento.

Existem alguns cenários nos quais aplicações necessitam armazenar remotamente muitas mensagens, em muito pouco tempo, como se estivessem relatando pequenos eventos para uma base ou um monitor, este tipo de aplicação encontra-se com facilidade na área de robótica ou vigilância. Para este cenários a comunicação de mensagens em sentido único tem demonstrado ser o modelo de comunicação canônico de melhor desempenho. A principal característica de funcionamento é o simples fato de emitir-se uma mensagem para um destinatário e continuar com o processamento sem que haja necessidade de dedicar tempo e esforço para a verificação sobre como e se esta mensagem atingiu o destino. Para melhor entendimento deste modelo de comunicação assíncrona de mensagem, oferecemos a seguir a Figura 4.4.4-9 comunicação de mensagem assíncrona em sentido único.

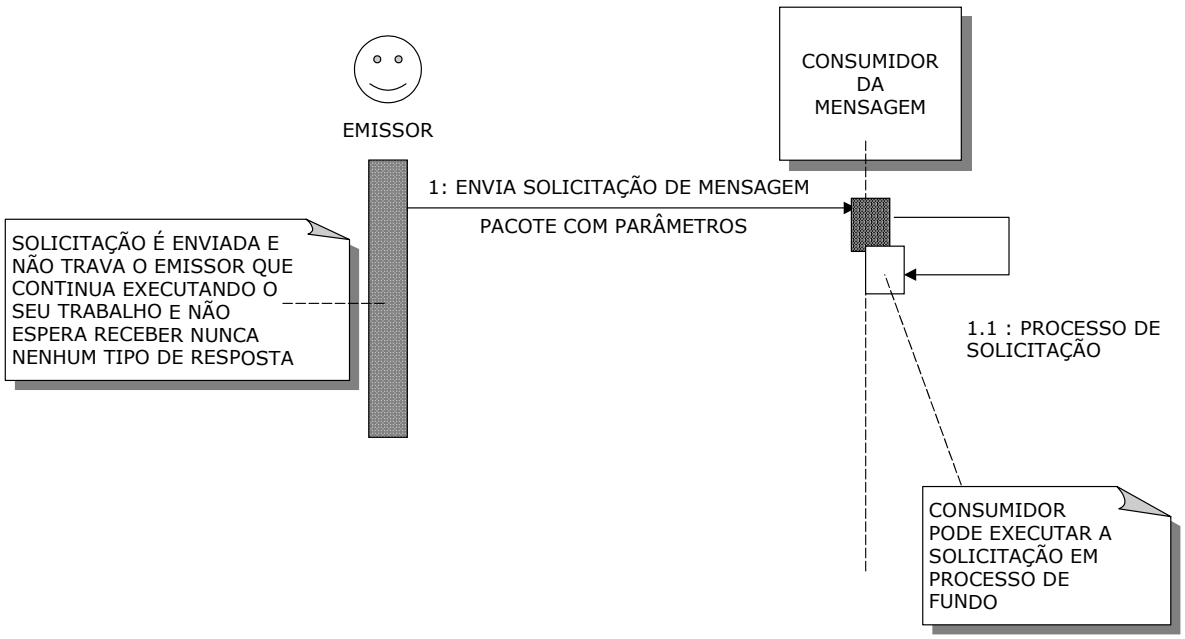


Fig. 4.4.4-9 comunicação de mensagem assíncrona em sentido único.

No modelo de aplicação de comunicação assíncrona faz-se necessário uma analogia com o modelo de servidor de correio eletrônico em Java Mail ou em qualquer outra plataforma [11]. Uma mensagem de correio eletrônico é enviada e pode ou não receber uma resposta ou uma confirmação de recebimento, dependendo dos detalhes que estejam informados na mensagem de correio eletrônico e como o sistema esteja configurado [11].

O emissor de uma mensagem ou o consumidor da mensagem (que pode ser um cliente ou um servidor) constituem-se muito mais percepções lógicas do que físicas sob o ponto de vista de entidades em uma arquitetura [11].

Uma aplicação empregando um provedor JAX M pode comporta-se em um cliente desempenhando o papel de um bom servidor, isto significa que ela possui a capacidade de realizar o chaveamento, a seleção, o direcionamento da mensagem entre o emissor e a mensagem do receptor ou consumidor [11].

Por outro lado uma aplicação JAX M que não utiliza a estrutura do provedor, será capaz de fornecer suporte somente para o cliente e enviar mensagens síncronas, como as aplicações que são denominadas clientes isoladas (standalone) de JAX M [11].

4.4.5 – Modelo síncrono de desenvolvimento JAX M

A especificação SAAJ e o pacote Javax.xml.soap definem objetos necessários para construir e desconstruir (fatorar) mensagens SAOP e enviá-las, como também recebe-las, sem que haja a necessidade do provedor. A API SAAJ fornece essencialmente ferramentas para a manipulação de documentos XML muito próximos ao estilo DOM que baseia-se em árvores de representação SOAP navegáveis pelos programas de desenvolvedores [11]. A seguir fornecemos passos principais, necessários no lado emissor, para que seja possível o envio ponto a ponto, síncrono de mensagem XML através da API JAX M empregando objetos e interfaces disponíveis nesta API :

- obter do ambiente MessageFactory para criar objeto mensagem;
- obter instância Message, a partir da message factory, obtida no item anterior;
- popular a mensagem SOAP, inserir conteúdos, montar a mensagem com os detalhes da regra de negócio e toda a informação relevante, incluindo-se aí os anexos necessários;
- obter do ambiente SOAPConnectionFactory que deverá fornecer objetos para conexão SOAP;
- obter instância SOAPConnection, a partir de SOAP connection factory do item anterior;
- criar Endpoint, ponto de acesso, o ponto de destino para a conexão;
- enviar a mensagem para o ponto de acesso, ponto de destino, empregando a conexão obtida e recebendo como retorno o SOAPMessage [11].

Esses passos referem-se ao transporte de mensagens SOAP que estarão disponíveis para manuseio e manipulação baseadas no estilo de representação em árvore semântica de documentos XML, como é intrínseco do estilo DOM [11]. Os participantes da comunicação deverão estar disponíveis e conectados no momento da troca de mensagens, é o modelo síncrono. Os passos aqui pontuados constituem-se tópicos, que englobam, cada um, grande quantidade de detalhes, os quais estão fora do escopo de pesquisa deste trabalho, porém podem ser facilmente obtidos no site da Sun Microsystems, nas referências [11] e [10] e no site http://www.webservicesarchitecture.com.

4.4.6 – Modelo assíncrono de desenvolvimento JAX M

Para este modelo de desenvolvimento assíncrono, temos de considerar a presença da infra-estrutura de apoio do provedor. Deste modo, o lado cliente que possui provedor, o qual comporta-se fazendo papel de proxy, aceitando e recebendo mensagens como um recipiente e

disponibilizando fachada assíncrona para a aplicação cliente [11]. Neste arranjo o cliente comporta-se de modo assíncrono e pode receber mensagens assíncronas a qualquer tempo dos servidores, dos consumidores de suas mensagens, praticando integralmente o modelo de comunicação de mensagens assíncrona.

A emissão de mensagens assíncronas, conforme definida no modelo da API JAX M, apresenta dois requisitos básicos que o serviço de envio cliente deve atender : deverá ser empregado o provedor JAX M e existir a camada perfil (profile) de mensagem diretamente conectada com ao provedor JAX M [11].

Em caso de dispensar o uso do provedor para a emissão e entrega de mensagens produzidas por si, o cliente simplesmente estará praticando o modelo de comunicação de mensagem síncrona como um cliente síncrono[11]. Como descreve a topologia MOM, em [10], não se faz condição necessária e indispensável que o cliente e o provedor estejam instalados e coexistam no mesmo endereço ou mesmo espaço físico compartilhando o mesmo hardware ou processo [11].

Por outro lado temos a restrição de que o provedor não pode expor-se isoladamente e de modo remoto, ou seja não é permitido que a API para a conexão com o provedor seja implementada sem a habilidade de realizar a conexão com o provedor através da rede. O cliente e o provedor devem manter coexistência mínima, obrigatoriamente lógica.

A especificação da API JAX M disponibiliza a solução para este requisito através da referência de implementação, pois o modo como uma aplicação de comunicação de mensagens assíncronas comunica-se com o seu provedor é diretamente definido pela especificação e implementação do vendedor e não imposto pela API JAX M [11] ou seja o provedor adotado deve responder aos padrões definidos pela API JAX M e pode estar em qualquer lugar onde a aplicação de comunicação de mensagens assíncronas desejar e puder conectá-lo para receber e para enviar mensagens.

Instalado e configurado provedor JAX M e perfil (profile) da mensagem com o provedor, no lado cliente, a emissão e o recebimento de mensagens assíncronas pela aplicação deve percorrer os seguintes passos fundamentais e gerais :

- criar uma conexão para o provedor, usando JAX M;
- empregar a conexão para o provedor para criar a mensagem a ser enviada;

- popular a mensagem SOAP, inserir conteúdos, montar a mensagem com os detalhes da regra de negócio e toda a informação relevante, incluindo-se aí anexos, se necessários;
- enviar a mensagem para o destino, porém através do provedor, o diferencial é que a mensagem, antes de seguir ao seu destino final, será enviada para o provedor [11], passando pelo perfil (profile) e neste estágio, do provedor, poderá ainda contar com facilidades inerentes a este modelo de comunicação com mensagens assíncronas, só disponíveis nele.

Aparentemente contendo menor número de passos e etapas para o envio de mensagens, o modelo de comunicação de mensagens assíncrono, encapsula, algumas características e detalhes que devem ser considerados e citados. Neste modelo temos um maior número de componentes, faz-se necessário a configuração e o ajuste da infra-estrutura de apoio do provedor e da comunicação deste com o cliente que deverá empregar a API JAX M. O outro aspecto a considerar é a implementação dos perfis (profile) de mensagens que devem ser ajustados podendo estar ligados a API JAX M ou diretamente ao provedor, a presença dos perfis (profile) deve ser considerada como alternativa para a solução do problema de correlação de mensagens, inerente ao modelo de comunicação assíncrona considerado.

Os pontos anteriormente mencionados constituem-se o fundamento para o envio de mensagens assíncronas em estruturas adequadas a este modelo. O escopo deste trabalho encerra-se na citação dos mesmos, porém faz-se necessário alertar para o fato de que a concretização desses, encontra-se recheada de detalhes e procedimentos menores e muito mais sofisticados que poderão ser executados mediante informações complementares e auxiliares encontradas em http://www.webservicesarchitecture.com, no site da Sun Microsystems e nas referências [11] e [10].

4.4.7 – JAX M e J2EE

A expansão da tecnologia de comunicação de mensagens empregando HTTP e SOAP nos modelos síncrono e assíncrono apresenta-se como possibilidade de disponibilização para outras tecnologias Java, como por exemplo J2EE.

Nas especificações relativas a versão J2EE 1.4 existem definições de como os message-drive EJB poderão realizar os serviços e funções desempenhados pela API JAX M, os quais por sua natureza de não possuir interfaces home, local home, remote ou local e estarem completamente desacoplados dos clientes que não podem acessá-los através de interfaces EJB. Com tais características

os message-drive EJB podem ser adaptados para receber mensagens e comportarem-se como servidores na camada de abstração que processará mensagens assíncronas no ambiente J2EE [11].

A possibilidade de emprego dos message-drive EJB no processamento de mensagens assíncronas no ambiente J2EE, leva em consideração a característica de que os mesmos não consideram e nem retém qualquer tipo de estado, tornando-os adequados para o emprego de estado não conversacional [11].

Dentro do ambiente J2EE podemos ainda dispor de mecanismos capazes de disponibilizar várias instâncias do bean que poderão processar múltiplas mensagens concorrentemente, agregando escalabilidade de volume ao processamento assíncrono de mensagens.

A implementação de message-drive EJB como Web Services que desempenharão as funções da API JAX M, terá como ponto de entrada servlets baseados em interfaces que, conforme o emprego, especificarão se o message-drive EJB estará processando uma mensagem assíncrona ou síncrona. Entre as vantagens de aquisição da tecnologia representada pela API JAX M para o ambiente J2EE, teremos a disposição do processamento de modelos de comunicação baseados em mensagens síncronas e assíncronas os benefícios e facilidades inerentes aos containers de EJB como por exemplo serviços de transação, segurança de aplicação, acesso heterogêneo a bancos de dados e outros [11].

4.4.8 – Interoperabilidade e JAX M

Entre duas partes que trocam mensagens empregando a tecnologia Web Services deve haver teoricamente o acordo universal da tecnologia Web Services que pressupõe, neste caso, mensagens enviadas e recebidas em SOAP e HTTP nos modos síncrono e assíncrono conforme acordos prévios descritos em WSDL.

Para que dois sistemas troquem mensagens de negócios empregando diferentes plataformas de hardware e softwares os quais desejam interoperar naturalmente, deverá haver acordo mínimo sobre os seguintes pontos fundamentais :

- especificar de modo formal o significado de pacote, publicação e a troca informações estruturadas e não estruturadas através da aplicação e dos limites das organizações, todos os documentos e mensagens trocadas devem ser bem definidos e bem entendidos entre ambos;

- realizar acordo que deve chegar ao nível do protocolo de comunicação da aplicação para realizar a comunicação da informação, detalhes de protocolo concreto para a troca de informações devem ser muito bem formalizados, entendidos e acordados entre as partes;
- disponibilizar mecanismos para a segurança de mensagens, integridade de mensagens, privacidade e não repudiação(significa registrar de modo confiável todas as tentativas de comunicação que uma parte tentou realizar com a outra e por algum motivo não conseguiu obter resposta para a troca de comunicação) [11] .

Devido ao fato das mensagens estarem estruturadas em XML, passado como forma padrão, segundo especificações baseadas em SOAP e empregando o protocolo padrão HTTP, emissores e consumidores de mensagens, podem ser um pouco, em algum nível, interoperáveis entre si [11]. Um serviço JAX M deve ser capaz de consumir mensagens SOAP 1.1 com anexos. O serviço fonte que envia a mensagem é desconhecido e pode ser produzido por qualquer aplicação empregando qualquer tecnologia [11]. Este é o princípio universal perseguido pela tecnologia Web Services, a interoperabilidade.

As partes que desejam comunicar-se devem concordar sobre a estrutura e entender a informação no contexto de negócios, incluindo-se a semântica de como estes documentos e informações devem ser trocados [11]. Este acordo ou combinação deve ser firmado sobre a camada de perfil (profile) que é o lugar indicado, pois os perfis (profiles) estão situados sobre os provedores [11]. Ao realizarem um acordo sobre como funcionará o perfil (profiles) e suas regras de negócio em detalhes, cliente e o serviço, não necessitam empregar o mesmo provedor ou serem escritos na mesma linguagem de software ou funcionarem sobre a mesma plataforma de hardware [11]. A única e mais forte de todas as exigências que deve permanecer é a de que o pacote de mensagem deve estar de acordo com o padrão SOAP [11]. No caso da API JAX M atuando no papel de cliente ou serviço, a interoperabilidade com outro serviço ou cliente, empregando diferentes provedores, requer que ambos os lados estejam ligados ao mesmo tipo de transporte, como no caso o protocolo HTTP, e o mesmo perfil (profile) configurado para ambos os lados na construção de mensagens SOAP [11].

4.5 – JAX R

Para comunicação de aplicações Java com registros de Web Services, denominados de UDDIs, foi disponibilizado conjunto padronizado de funções agrupadas na API JAX R (Java API for XML Registries) que permite a interação com estes mecanismos, para uso em registro/publicação e

descoberta/busca de Web Services, a partir de registros baseados em XML, tais como UDDI e ebXML Registro/Repositório [10]. A API JAX R constitui-se parte integral da plataforma J2EE 1.4 disponibilizada a partir de 2003 [10].

As definições de JAX R não incluem especificação para nenhum tipo novo de registro a ser adotado como padrão [10] ao contrário, o objetivo da API JAX R é definir padrões para a API Java executar operações de registro sobre a maior diversidade possível de conjuntos e modelos de mecanismos de registros para Web Services [10].

A API JAX R pretende interpor-se como camada entre aplicações Java, no segmento de Web Services , que realizam interação e gerenciamento de registros para Web Services [10].

Considerando que os vários registros para Web Services possuem características diferentes no que diz respeito a modelo de informação e funcionamento. Esta padronização implementada por JAX R tem como função prática permitir a troca de registros, com o mínimo de impacto sobre a aplicação Java que interage com os mesmos. Um dos caminhos para atingir este objetivo é através da unificação de vários modelos de informação, já estabelecidos no segmento de Web Services, através da interface JAX R, o que poderá possibilitar que mesmo com a troca ou a diferenciação do registro em uso, o mesmo código da aplicação Java possa continuar a ser usado para o gerenciamento do registro[10], obviamente, sofrendo muito pouco impacto, no sentido de utilizar as características relevantes do novo registro agregado ao diálogo de interação entre aplicação e estrutura de registro.

Para atingir os objetivos e a funcionalidade anteriormente exposta, JAX R apoia-se em uma complexa rede semântica, composta por elementos extensos e detalhados de arquitetura e sobre modelo de informação altamente flexível, de baixa granularidade que tenta adequar-se aos muitos modelos de informação para registo de Web Services já estabelecidos neste segmento.

4.5.1 –Arquitetura JAX R

A arquitetura da API JAX R possui três componentes principais : provedor de registro, provedor de JAX R e cliente JAX R [10]. A seguir pontuaremos cada um destes componentes :

- provedor de registro - constitui-se na camada de implementação dos parâmetros de registos a serem usados como UDDI, ebXML registro/repositório ou registro OASIS. Um provedor de

registro poderá ou não disponibilizar o implementação de um registro para o JAX R [10]. Este componente representa a característica de cada modelo de informação que a API JAX R poderá manipular. Se algum outro modelo tornar-se disponível e sua inclusão se fizer necessária no padrão estabelecido pelo JAX R, o componente de arquitetura provedor de registro deverá ser alterado para comportar esta inovação;

- provedor JAX R – implementa a especificação JAX R. Os outros componentes da arquitetura como as aplicações JAX R e os clientes JAX R devem acessar e interagir com o registro desejado através da camada/componente provedor JAX R. Existem três categorias de provedor JAX R : provedor de encaixe (plugable), provedor especifico para registro e provedor JAX R ponte. No caso dos provedores de encaixe estes tem como características principal a possibilidade de poder ser encaixados a qualquer registro com os quais sejam compatíveis e adequados, geralmente carregam dentro de si a possibilidade de ser utilizados com vários tipos de registro. Os provedores específico para registro constituem a categoria de provedores personalizados escritos para determinado tipo de registro que através deles pode ser manipulado pela API JAX R, constituem-se em uma espécie de personalização. Provedores JAX R ponte são específicos para modelos já estabelecidos no segmento de Web Services e otimizados para um dos padrões como UDDI ou ebXML registro/repositório de modo que independem do vendedor e estão fortemente voltados para o padrão do registro e da API JAX R [10];
- aplicação JAX ou cliente JAX R – uma aplicação Java que interage ou acessa um registro de Web Services em um diretório que expõe este tipo de conteúdo, deve ser considerada um cliente JAX R, pois para isso necessita empregar esta API [10] entrando em contato com o provedor JAX R. Esta aplicação cliente pode ser um programa em Java funcionando isolado (standalone) em um computador ligado a uma rede que tenha acesso a um registro de Web Services, um componente de uma sistema corporativo, como um EJB ou um servlet, a partir de um container servlet [10]. Tipicamente, a experiência tem demonstrado que um cliente JAX R e um provedor JAX R compartilham e residem na mesa JVM (Java Virtual Machine) [10].

A Figura 4.5.1-1 arquitetura JAX R, apresenta esquema representativo da arquitetura JAX R e seus componentes, incluindo-se ai os níveis de interação entre os mesmos, os aspectos intercaláveis entre as partes e a diversidade entre os vários tipos de registros que o padrão JAX R pretende interagir.

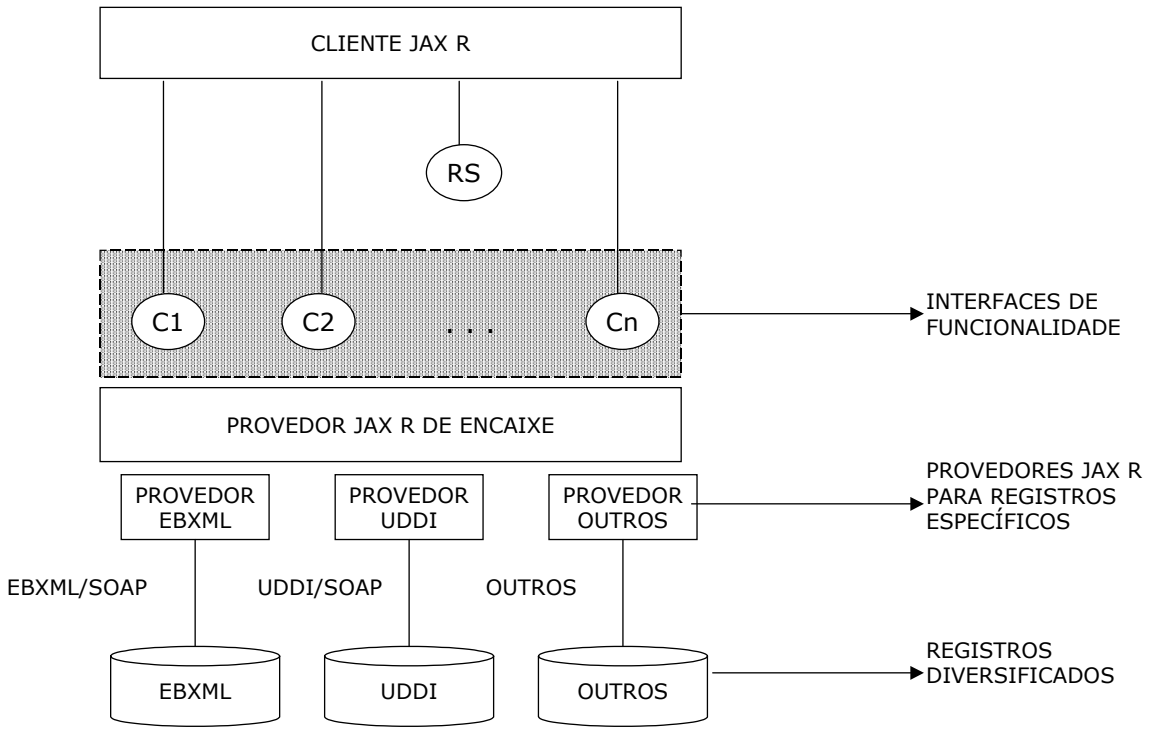


Fig. 4.5.1-1 arquitetura JAX R

Na figura anterior podemos observar o RS (Registry Service) que representa o serviço de registro, a principal interface da arquitetura que deve ser implementada por todos os provedores JAX R. A partir do RS, uma aplicação cliente estabelece conexão com o registro via interface Connection que poderá, dependendo da implementação e da condução do processo, estabelecer as funcionalidades para a interação com resgistro [10]. A interface Connection esta vinculada a parte da API JAX R que realiza o gerenciamento de conexão.

Os círculos C1, C2 até Cn representam, na figura anterior, interfaces implementadas pelo provedor JAX R . Cada uma destas interfaces tem como responsabilidade uma funcionalidade específica de um dos tipos de registros, como busca de dados do registro, adição, modificação e exclusão de dados dos registros e dos próprios registro em si [10]. Na terminologia adotada pela API JAX R estas interfaces recebem a denominação de interfaces de funcionalidades (capability interfaces) [10].

4.5.2 – Funcionalidades JAX R e perfis (profile) de funcionalidades

Como mencionado anteriormente, a API JAX R tem como objetivo disponibilizar uma API simples e unificada (um padrão mínimo) para acesso e interação com diferentes registros. Entretanto, os modelos, estruturas e funcionalidades dos registros disponíveis variam

significativamente, principalmente no conjunto de opções funcionais e na organização interna dos seus modelos de informação [10].

Ao implementar uma API para suportar maior número possível e o mais diversificado conjunto de funcionalidades, a tendência é que a implementação seja direcionada para os conjuntos padrões de opções e funcionalidades que estejam presentes na grande maioria dos registros em detrimento das funcionalidades específicas presentes apenas em alguns ou em pequena minoria[10].

O grau de variação de funcionalidades que são providas por registros diferentes pode ser considerada bastante grande, iniciando-se pela adoção de protocolos completamente fora dos padrões e estendendo-se ao modelo de informação adotado e escolhido como suporte[10]. Neste contexto, o projeto de uma API que tenta a unificação, como é o objetivo de JAX R, tem de considerar em tornar o ponto focal no máximo utilizável para um e para outro tipo de registro, mesmo considerando a adversidade de ambos [10].

Neste contexto, o principal objetivo da API JAX R é disponibilizar suporte para registros diversos e diferentes através de uma API que possa atingir todos os lados do escopo e que seja funcionalmente capaz de vir a torna-se um denominador comum[10] em termos de convergência para uma API padrão que cubra a maioria das funcionalidade consideradas realmente necessárias e essenciais.

Para ser capaz de manter a oferta de alta e variada funcionalidade em grupos específicos e bem definidos e ao mesmo tempo não incorrer no equívoco de tornar a API uma estrutura que sofresse inchaço desnecessário a cada inclusão de nova funcionalidade ou padrão de registro, JAX R introduziu o conceito de facilidades funcionais e seus perfis [10] que serão pontuadas a seguir com as denominações de funcionalidades e perfis de funcionalidade :

- funcionalidades – constitui-se em um grupo relativo a operações que podem ser realizadas nos registros e em suas partes, entre estas a busca de um registro, a adição de novos registros e outras. Cada funcionalidade representa uma interface Java no conjunto JAX R. Entretanto devemos considerar que o modelo de implementação emprega o conceito de agrupamento de funcionalidades, segundo seu desempenho em relação ao registro e agrupa funcionalidades similares de acordo com os níveis de aplicações e com os níveis de uso;
- perfis de funcionalidades – um perfil de funcionalidade é um grupo de interfaces Java que encontra-se no mesmo nível. JAX R define dois níveis de perfis diferentes para agrupar funcionalmente as funcionalidades que mesmo realizando operações semelhantes (busca, adição, remoção de registros

e outros) estarão agrupadas em perfis diferenciados. Estes perfis dividem-se em dois : básico e avançado [10].

A partir da estrutura em dois níveis : básico e avançado, os perfis de funcionalidades agrupam interfaces para a execução de tarefas semelhantes, porém com aspectos de complexidade bastante distintos entre si e sobre elementos pertencentes a domínios distintos :

- perfil básico, nível 0, disponibiliza suporte básico para o gerenciamento do ciclo de vida e para as funções de busca e pesquisa de registros. Interfaces que pertencem ao nível 0 do perfil de funcionalidade são denominadas como interfaces de negócio (business). No nível 0 do perfil de funcionalidades temos as interfaces gerenciamento do cliclo de vida de negócio (BusinessLifeCycleManager) e gerenciamento de busca de negócio (BusinessQueryManager);
- perfil avançado, nível 1, neste perfil temos suporte avançado para os aspectos funcionais de gerenciamento do ciclo de vida e das funcionalidades de busca e localização. Estas funções implicam no suporte disponibilizado para o nível 0, incluindo funções avançadas e flexibilidades que podem ser empregadas somente neste nível. As interfaces que pertencem a este nível são denominadas genéricas (generic). Entre as interfaces que fazem parte deste nível temos gerenciamento de ciclo de vida (LifeCycleManager) e gerenciamento declarativo de busca (DeclarativeQueryManager) [10].

Para registros que não tem suporte aos padrões disponibilizados pelo perfil básico de funcionalidades, nível 0, a API JAX R disponibiliza acesso a estes registros através do perfil avançado, nível 1, de implementação que pode ser utilizado para acesso simplificado ou para implementação de perfil mais avançado e mais adequado ao registro, se for o caso[10] . Como funcionalidade adicional uma aplicação JAX R pode inferir junto a um registro qual o nível de complexidade do perfil de funcionalidades que mais se adere as suas características através da interface perfil de funcionalidade (CapabilityProfile) [10].

4.5.3 – Modelo de programação JAX R

Mesmo estando sobre um complexo modelo de informação e gerenciando conjunto vasto e detalhado de interfaces que cobrem cada detalhe e escopo do modelo de informação e do objetivo de tornar-se padrão de API entre registros, a API JAX R possui modelo de programação relativa mente simples e compacto, composto de dois pacotes principais :

- Javax.xml.registry - este pacote define as interfaces responsáveis pela disponibilização usual de serviços de registros que são objetos fundamentais em funcionalidades como gerenciamento de conexão, gerenciamento de ciclo de vida e gerenciamento de consulta e busca de registro [10]. Registro de serviços é responsável pela parte inicial, o contato com o registro, todas as demais funcionalidades básicas fluem a partir dele e através dele.
- Javax.xml.registry.infomodel – o modelo de informação é representado por este pacote. A manipulação, a implementação e a definição do modelo de informação sobre o qual a API JAX R atua encontra-se neste pacote [10]. Todos os elementos e entidades de informação estão definidos neste pacote e são passados pela API JAX R para os registros de acordo com a adequação destes.

4.5.4 – Modelo de informação JAX R

O termo modelo de informação refere-se ao tipo de informação que pode ser armazenado, suportado e mantido por um diretório ou registro em particular. Considerado como uma das partes mais importantes de um registro, o modelo de informação tem como função tornar-se o mais rico possível em termos de informação, o que tornará a utilização do registro muito maior, mais interessante e mais relevante na procura do Web Services que o cliente estiver interessado[10].

Registros diferentes estão assentados sobre diferentes modelos de informação. De fato, não há nenhuma padronização sobre modelos de informação para registros e diretórios de exposição de Web Services baseados em XML [10]. A API JAX R constitui-se o primeiro padrão na tentativa de disponibilizar regras e orientações para unificar a visão do modelo de informação para registros baseados em XML[10] que exponham e publiquem indicações de Web Services.

A base para o modelo de informação empregado por JAX R deriva do modelo de informação do registo ebXML conforme definido na especificação do ebXML registro de modelo de informação (registry information model RIM). Entretanto, este modelo representa apenas a base, o mesmo foi estendido para suportar e adequar-se ao padrão UDDI [10] que representa um dos grandes padrões já estabelecidos e amplamente empregados no segmento de Web Services.

O modelo ebXML RIM representa um modelo de informação amplo, de simples entendimento, para informação estruturada de registro, se comparado com modelos como UDDI estrutura de dados (UDDI Data Structure UDDI-DS), o qual JAX R disponibilizou suporte por, herança, proveniente do modelo ebXML RIM desde o início de sua implementação [10].

As interfaces relativas ao modelo de informação gerenciado por JAX R estão definidas em um pacote diferente denominado Javax.xml.registry.infomodel. Estas interfaces representam a ligação entre a linguagem Java e o modelo de informação unificado para duas especificações dominantes na tecnologia de registros para Web Services que são : UDDI e registro ebXML [10]. É importante salientar que o modelo de informação gerenciado pela API JAX R, representa tão somente uma visão de como a informação deve ser estruturada em registros compatíveis e baseados em JAX R e não representa a estrutura que modelo de informação do repositório em questão mantém, em sua constituição física [10].

4.5.5 – Classes e interfaces do modelo de informação JAX R

O modelo de informações gerenciado pela API JAX R compõem-se de dezenove classe e interfaces, consideradas de certa importância e uso freqüente [10]. A complexidade de uso destes componentes esta diretamente relacionada com o modelo de informação do registro com o qual a aplicação Java, baseada em JAX R vai interagir. A seguir pontuaremos os principais componentes :

- registro objeto (RegistryObject) – constitui-se em classe base que estende a maioria dos objetos no modelo de informação JAX R. Este objeto disponibiliza conjunto mínimo de metadados sobre objetos disponíveis no registro [10]. É a classe mãe de todas as classes que compõem o modelo de informação;
- registro de entrada (RegistryEntry) – esta interface destina-se a suportar requisitos adicionais, informações complementares de outros objetos com relação a fina granularidade de informações (detalhes) de modos mais específicos e freqüentes, necessários para descrever aspectos de objetos mais importantes;
- organização (Organization) – o tipo organização estende a registro objeto com o objetivo de conter toda e qualquer informação sobre organizações presentes no modelo de informação baseado em JAX R;
- serviço (Service) – representa a informação pertinente ao Web Services que está publicado pela organização no registro. Constitui-se também uma extensão do registro objeto, podem haver zero ou mais serviços e as ligações são representadas pelos objetos ligações de serviço;
- ligação de serviço (ServiceBinding) - estende a instância de registro objeto. Está relacionado com o serviço publicado pela organização e disponibiliza informações técnicas referentes aos mecanismos de acesso como URI, disponibilizando interfaces e modos para acesso ao Web Services;

- especificação de link (SpecificationLink) – para cada ligação de serviço deve haver uma especificação de link indicando como informações técnicas podem ser obtidas de como pode-se invocar um Web Services SOAP RPC ou outras relativas a outros contextos;
- esquema de classificação (ClassificationSchema) – o modelo de informação JAX R pode ser usado para especificar taxionomias próprias e particulares na classificação de registro objeto, bem como adotar esquemas de classificação já estabelecidos. Esta interface pode ser empregada para criar esquemas de classificação do modelo de informação;
- classificação (Classification) – podemos classificar registro objetos em vários grupos e instâncias, para tal temos a interface classificação que está ligada a algum esquema de classificação, tornando o modelo de informaçõ JAX R ainda mais flexível;
- conceito (Concept) – pode ser empregado para representar qualquer coisa virtualmente. Alguns dos usos mais comuns desta classe são : pode ser usado para definir a arvore hierárquica de classificação em um esquema e pode ser usado como proxy para conteúdos armazenados externamente em registros nível 0 com identificador único;
- associação (Association) – este tipo é utilizado para definir a associação entre objetos diferentes no modelo de informação gerenciado pela API JAX R;
- identificador externo (ExternalIndetifier) – as instâncias de identificadores externos são empregadas para produzir informação de identificação para objetos de registro em chaves de 128-bit UUID;
- link externo (ExternalLink) – instâncias deste tipo disponibilizam ligações, apontadores URI que são gerenciados fora do registro;
- slot (Slot) – uma instância slot possibilita a adição arbitrária de atributos às instâncias de registro objeto em tempo de execução;
- objeto extensível (ExtensibleObject) – esta interface consiste em métodos que permitem a adição, exclusão e procura de instâncias slot agregadas a objetos registro;
- objeto extrínseco (ExtrinsicObject) – tipo empregado para disponibilizar metadados sobre os itens do repositório, os quais o registro do repositório não tem a informação, algo como a informação de origem dos itens do repositório;
- evento auditável (AuditableEvent) - instância do objeto registro empregada para registrar tudo que ocorre, fazer um log, uma trilha de tudo que acontece com um determinado objeto registro;
- usuário (User) – instância do objeto registro empregada para registrar informações cadastrais sobre usuários registrados para interagir e manipular com o modelo de informações mantido pelo registro;

- endereço postal (PostalAdress) – representa o endereço postal empregado como parte de usuário e organização;
- pacote de registro (RegistryPackage) – empregado para agregar logicamente grupos relacionados de instâncias de objetos registos [10].

Existem possíveis relacionamento entre os dezenove componentes que são empregados de acordo com situações de aplicação no que se refere aos aspectos de segurança da informação, classificação e agregação de similaridades [10].

4.5.6 – Classificação de registros objetos

O modelo de classificação de objetos registros permitido pela API JAX R demonstra-se bastante flexível e apoia-se em conceitos reais de classificação. Classificação nos remete a categorização de entidades baseadas em esquema bem definido de classificação ou taxionomia. A classificação possibilita a rápida localização do registro desejado [10].

A API JAX R suporta os mais diversos meios e métodos de classificação, tendo como característica grande capacidade de busca e recuperação de registros baseados nestes esquemas [10].

Os principais elementos empregados no esquema de classificação do modelo de informação gerenciado pela API JAX R são três interfaces : classificação, esquema de classificação e conceito. A interface classificação é empregada para classificar o registro objeto que pode ser classificado em zero ou várias classificações e pode ser incluído e excluído de classificações a qualquer tempo [10].

A interface esquema de classificação é utilizada para representar taxionomias ou esquemas maiores de classificação que agregam as classificações ligadas diretamente aos objetos registro. Uma classificação utiliza o esquema de classificação para identificar a taxionomia empregada para classificar determinado registro objeto em uma dimensão em particular [10]. E um objeto registro pode ter várias classificações e pertencer a várias taxionomias, porém uma classificação só poderá pertencer a um único esquema de classificação. Ao pertencer a várias classificações e portanto a várias taxionomias o registro objeto pode ser denominado, quanto a este aspecto, de classificação multidimensional [10].

Taxionomias são definidas em termos de estruturas constituídas por elementos, denominados elementos de taxionomia e os relacionamentos destes elementos entre si. A API JAX R suporta as características relativas a taxionomias reais baseada em dois modos na localização dos elementos e da estrutura de taxionomia e na estrutura de relacionamento da informação que podem ser : internamente e externamente [10]. No caso da taxionomia interna, os elementos da taxionomia e suas estruturas de relacionamento estão disponíveis internamente no provedor JAX R. Este tipo de taxionomia disponibiliza mais funcionalidade e valor agregado para a aplicação JAX R devido ao fato de que o provedor JAX R ter a possibilidade de validar concretamente as referências aos elementos da taxionomia em uma classificação, adicionando maior significado e correção [10]. Entretanto, a estrutura interna mantida com o provedor JAX R necessita ser atualizada sempre que a taxionomia evoluir, o que representa esforço de manutenção que deve ser considerado na adoção deste tipo de taxionomia. Entretanto, esta modalidade torna a taxionomia interna muito mais flexível para mudanças. [10]. Todo o trabalho de manutenção de taxionomia a ser realizado pela API JAX R deve ser executado através do conceito de interface que gerência toda a árvore semântica de esquema de classificação e suas classificações associadas, requerendo esforço de programação e de entendimento de classificação.

Nas taxionomias externas, informações sobre elementos da taxionomia e suas estruturas estão disponíveis externamente ao provedor JAX R que podem utilizar estas taxionomias externas sem ter de importar a estrutura completa destas. Como a estrutura completa reside fora do provedor JAX R, não há necessidade de se atualizar a taxionomia toda vez que esta evolui ou sofre novos acréscimos [10]. A desvantagem de taxionomias externas consiste na impossibilidade para que o provedor JAX R possa validar as referências para elementos de taxionomia nas classificações [10].

As classificações de um registro estão sempre ligadas a esquemas de classificação que representam taxionomias. Em qualquer dos casos a instância classificação sempre deverá empregar e está ligada ao esquema de classificação para identificar a taxionomia da qual faz parte, se externa ou interna. Considerando ambas possibilidades de tipos de taxionomia que uma classificação pode estar ligada, temos a citar classificações internas e externas.

Para instância classificação empregada para classificar um objeto registro usando uma taxionomia interna, a classificação é denominada classificação interna relacionada a uma instância conceito que representa o elemento da taxionomia posicionando este elemento de classificação em relação a raiz do esquema de classificação interno[10].

Em outra situação a instância classificação é empregada para classificar o objeto registro referenciando uma taxionomia externa, a classificação é referenciada como uma classificação externa. A

API JAX R possui mecanismos para atribuir os valores adequados a esta classificação, mas não há nenhuma checagem em relação a validações externas da taxionomia a qual a classificação externa faz referências [10] todo arranjo fica baseado em função dos valores atribuídos internamente ao registro.

4.5.7 – Associação de registros objetos

As associações entre os registros de objetos são inerentes ao modelo de informação suportado pela API JAX R. Uma instância de registro objeto pode ser associado a zero ou mais instâncias de registros objetos [10]. O mecanismo para expressar associações no modelo de informação JAX R é a interface associação, que pode se empregada para associar quaisquer dois objetos registro. Uma instancia da associação representa concretamente a associação entre um objeto registro fonte e um objeto registro alvo, denominados objeto fonte e objeto alvo, respectivamente [10]. As associações promovidas pela API JAX R guardam semântica quanto ao seu estabelecimento, com a denominação das origens, sempre é possível determinar qual é a origem e quem é o destino ou ainda quem é a fonte e quem é o alvo da associação [10].

Cada instância de associação possui o atributo associação tipo (associationType) que identifica o tipo de associação [10], adicionando mais um componente semântico para as relações do modelo de informação. JAX R define conjunto de tipos enumerados de associações que inclui todos as possíveis associações bem formadas entre dois objetos registros : relacionado a (RelatedTo); contém (Contains); substituir (Supersedes); tem parte (HasMember); equivalente (EquivalentTo); usa (Uses); tem fllho (HasChild); estende (Extends); troca (Replace); tem pai (HasParent); implementa (Implements); responsável (ResponsibleFor); ligação externa (ExternalLink); instância (InstanceOf); submisso (SubmitterOf). Estes são os principais tipos de relacionamentos disponíveis no modelo de informação gerenciado pela API JAX R que podem ser empregados na associação de instâncias registros de objetos [10].

As associações do modelo de informação entre registros objetos possuem pelo menos dois casos de uso bem definidos. Baseados nos participantes e elementos criadores das associações entre registros objetos e de como e para onde partiram as associações, podemos ter os seguintes casos de uso : associações intramurais e associações extramurais. Quando um usuário, dono dos dois registros objetos cria uma associação entre estes temos uma associação intramural que não precisa ser confirmada e por isso é efetivada automaticamente [10].

Entretanto, as associações extramural são um pouco mais sofisticadas, pois os usuários donos dos objetos podem ser diferentes entre si e também o usuário que está criando a associação pode não ser o proprietário de nenhum dos dois objetos. Neste contexto, a associação entre objetos deve ser confirmada pelos donos dos objetos que, opcionalmente, também podem não confirma-la[10]. Essa confirmação pode não se dar de modo instantâneo, pois um usuário pode propor a associação, a mesma fica em suspenso, aguardando que o dono ou os donos dos registros objetos se conectem ao provedor JAX R e confirmem ou não a mesma.

4.5.8 – API Registro de Serviço (Registry Service) do JAX R

A interface de registro de serviço (RS) pertencente a JAX R esta definida no pacote Java separado, chamado Javax.xml.registry. Como trata-se de uma interface de programação possui uma estrutura de montagem orientada em relação aos serviços que deve executar, agrupando funções por similaridade de uso de modo que temos três grandes grupos de serviços : gerenciamento de conexão (Connection Management API), gerenciamento do ciclo de vida (Life-Cycle Management API) e gerenciamento de busca/consulta (Query Management API).

O gerenciamento de conexão com o registro constitui-se em um grupo de atividades e funcionalidades menores que serão pontuadas a seguir de modo a expressar como RS fornece suporte a conversação, ao fluxo entre a aplicação baseada em JAX R e o registro. Considerando-se, obviamente, a intermediação do provedor JAX R em todas as ocasiões :

- obtendo uma fábrica para conexão (looking up for connectionfactory) – para que se possa conectar com um registro, através do provedor JAX R, faz-se necessário a obtenção inicial de uma fabrica de conexão. Existem dois meios para se obter uma fábrica de conexção : JAX R pode obter uma fabrica de conexão configurada e disponível no JNDI (Java Naming and Directory Interface) que deve estar registrada especificamente para JAX R. Ou através do método estático newInstance ( ) que pode indicar que fabrica de conexão deve ser instanciada quando chamado, a partir da configuração do arquivo system property;
- configurando propriedades de conexão na fabrica de conexão (setting connection properties on connectionfactory) – para que ocorra a efetiva comunicação com o registro, via provedor JAX R, faz-se necessário a configuração de propriedades adequadas ao tipo de registro que se esta empregando. As propriedades podem ser configuradas de forma padrão ou não padronizadas conforme propriedades especificadas pelo provedor JAX R;

- criando conexão JAX R (creating a JAX R connection) – a criação da conexão é realizada a partir da instância fábrica de conexão com a emissão do método createConnection ( ) que verifica se todas as propriedades adequadas a conexão e necessárias ao funcionamento da instância de gerenciamento de consulta estão definidas de modo consistente e sem discrepâncias entre si. JAX R suporta dois tipos de conexão : síncrona e assíncrona. Na conexão síncrona o provedor JAX R deve processar completamente cada método de solicitação chamado de modo síncrono antes de retornar uma resposta não nula JAXRResponse que deverá ser preenchido com o conteúdo solicitado. A aplicação baseada em JAX R, que chama o provedor sincronamente, fica bloqueada aguardando a resposta. Na comunicação assíncrona, o provedor JAX R, produz um identificador universal ID para a solicitação e devolve ao solicitante como uma instância de JAXRResponse. Processa internamente a solicitação e aguarda nova conexão do solicitante, que deve requerer mensagens assíncronas disponíveis para ele. Ao receber as mensagens assíncronas o solicitante deve proceder da seguinte forma com as instâncias de JAXRResponse : primeiro verifica se existe conteúdo referente a alguma solicitação de ID ou se a resposta consiste apenas na informação de um ID que terá seu conteúdo respondido futuramente [10]. Existe a possibilidade da aplicação baseada em JAX R trocar de modo de comunicação entre assíncrona e síncrona em tempo de execução [10];

- especificando credenciais de segurança (security credentials specification) – para algumas operações sobre o conteúdo dos registros como inclusão, exclusão, alteração de conteúdo e outros faz-se necessário credenciais privilegiadas que necessitam ser informadas por métodos específicos. Esta modificação de credenciais pode ser feita dinamicamente em tempo de execução, quando a operação assim o exigir. Para operações de consulta estas credenciais não se fazem necessárias;

- empregando a conexão para acessar o registro (using connection to acess the registry) – após o estabelecimento da conexão com o registro a interface RegistryService pode ser empregada para que a aplicação baseada em JAX R tenha acesso a funções como : gerenciamento de funcionalidades do ciclo de vida do registro através do provedor JAX R para criar, alterar, deletar objetos no registro alvo, através de métodos específicos. Poderá também ter acesso ao gerenciamento a funcionalidades de busca e recuperação de registro, também através de métodos específicos;

- encerrando a conexão JAX R (closing a JAX R connection) – a aplicação baseada em JAX R pode fechar a conexão com o provedor JAX R através da chamada de método específico. Este deve ser o procedimento padrão quando não se desejar interagir com o registro alvo;

- acessando múltiplos registros (accesing multiple registries) – dentro do caracter de disponibilidade da interligação de registro formando uma federação de registros, aplicações baseadas em JAX R podem disponibilizar acesso através dos modos federado e não federado. No modo federado a

conexão passa a ter caráter federado, onde uma simples conexão está ligada a vários registro ao mesmo tempo, este tipo de conexão está disponível exclusivamente para a distribuição de consultas distribuídas ou federadas entre os vários registros que devem comportar-se como registros simples na montagem da resposta. No modo não federado, a aplicação baseada em JAX R realiza várias conexões simultâneas, porem isoladas e individuais, com vários registros ao mesmo tempo e acessa cada registro como se fosse único realizando as operações que considerar adequadas [10].

A parte de JAX R que encarrega-se do gerenciamento de ciclo de vida constitui-se de uma API composta por duas interfaces :

- gerenciamento de ciclo de vida (lifecyclemanager) – esta interface disponibiliza suporte completo para a manipulação do ciclo de vida de todos os objetos que compõe e fazem parte do modelo de informação gerenciado pelo JAX R. Esta interface pertence a API genérica e possui suporte para tarefas como criação, modificação, exclusão, desativação/reativação de registros objetos;
- gerenciamento de ciclo de vida de negócios (businesslifecyclemanager) – suporte completo para manipulação do ciclo de vida dos objetos que são considerados chaves e inerentes a descrição do negócio, da organização, pertencentes a padrões estabelecidos como UDDI e ebXML RIM que fazem parte do modelo de informação gerenciado pelo JAX R, é fornecido por esta interface que pertence a API negócio e disponibiliza mecanismos para tarefas como : confirmação de associação, exclusão de associação, exclusão de esquemas de classificação, exclusão de conceito, exclusão de organização, exclusão de ligações de serviço, exclusão de serviço, inclusão de associação, inclusão de esquema de classificação, inclusão de conceito, inclusão de organização, inclusão de ligações de serviço, inclusão de serviço e não confirmação de associação. Todos estes mecanismos poderão atuar respectivamente sobre os seguintes objetos chaves do modelo de informação : organização, serviço, ligações de serviço, conceito, associação e esquema de classificação [10].

As operações de gerenciamento de ciclo de vida não são suportadas, em nenhum caso, sobre conexões federadas. Em caso de tentativa de realização de qualquer destas operações via conexão federada como provedores JAX R, ocorrerá uma exceção [10]. A grande maioria das operações de gerenciamento do ciclo de vida dos objetos é realizada em lotes, ou seja, pode-se passar coleções de vários objetos da mesma classe para que sejam inclusos, excluídos e etc [10].

O gerenciamento de pesquisa, procura e busca de registros em provedores JAX R feito por programas baseados em JAX R é disponibilizado por duas interfaces :

- gerenciamento de consultas de negócio (businessquerymanager) – esta interface disponibiliza funcionalidades para a consulta de objetos considerados chaves que fazem parte do modelo de informação gerenciado pelo JAX R. Pertencendo a API negócio, possui o seguinte conjunto de objetos considerados chaves no modelo de informação : organização, serviço, ligações de serviço, conceito, associação, esquema de classificação e pacote de registro [10]. Para atuar realizando buscas e procuras sobre estes objetos chaves, que podem estar relacionados ou não, estão disponíveis os métodos : procura associações, procura associação por formador, procura esquema de classificação por nome, procura esquema de classificação, procura conceito por caminho, procura conceito, procura organização, procura pacotes de registros, procura ligações de serviço, procura serviço. Os métodos anteriores possuem um conjunto de argumentos comuns muito bem definidos em suas aplicações e semântica sobre os objetos e atributos do modelo de informação. O grupo de argumentos comuns é constituído por : nomes de padrões, qualificadores, classificações, especificações, identificadores externos e ligações externas. A combinação dos atributos dos objetos chaves com a flexibilidade presente no conjunto de argumentos listados permite a interface alto poder de localização e busca de qualquer informação desejada sobre o modelo;

- gerenciamento de consultas declarativas – para suportar consultas a qualquer objeto que faça parte do modelo de informação gerenciado pelo JAX R, esta interface disponibiliza mecanismos de procura *ad hoc*, pois pertence a API genérica. Esta interface disponibiliza a possibilidade de realizar consultas *ad hoc* empregando linguagens de sintaxe declarativa. Linguagens como SQL-92 e OASIS ebXML Registry Filter Queries. Entretanto o suporte a estas linguagens é opcional dependendo do registro com o qual se está interagindo, em caso do registro não suportar a linguagem declarativa, ocorrerá uma exceção. Para utilização de linguagens de consultas declarativas estão disponíveis dois métodos : criação de consulta (createQuery) e o execução de consulta (executeQuery) [10].

Nestes três grupos bem definidos de API temos as funções necessárias para a interação de programas aplicativos baseados em JAX R com provedores JAX R, entretanto devemos considerar a diversidade dos registros e suas principais particularidades. A interação com registros baseados em padrões definidos e estabelecidos como UDDI-DS e ebXML RIM torna-se mais fluente e simplificada através do emprego canônico dos mecanismos disponibilizados pela parte negócio da API JAX R. Porém, para os casos de novos padrões ou de registros com particularidades que justifiquem ser exploradas e usadas o emprego de JAX R também é recomendável através da expansão da parte genérica da API que se encontra disponível.

4.6 – Conclusão

As API Java aqui bordadas referem-se às tecnologias do núcleo de Web Services. Apenas com a aplicação de JAX P, JAX RPC, JAX M e JAX R o desenvolvedor Java tem a possibilidade de consumir e produzir Web Services na Internet de modo natural e sem entrar em contato direto com os detalhes desta tecnologia. Na Figura 4.6-1, apresentamos graficamente a equivalência entre as tecnologias Java e Web Services.

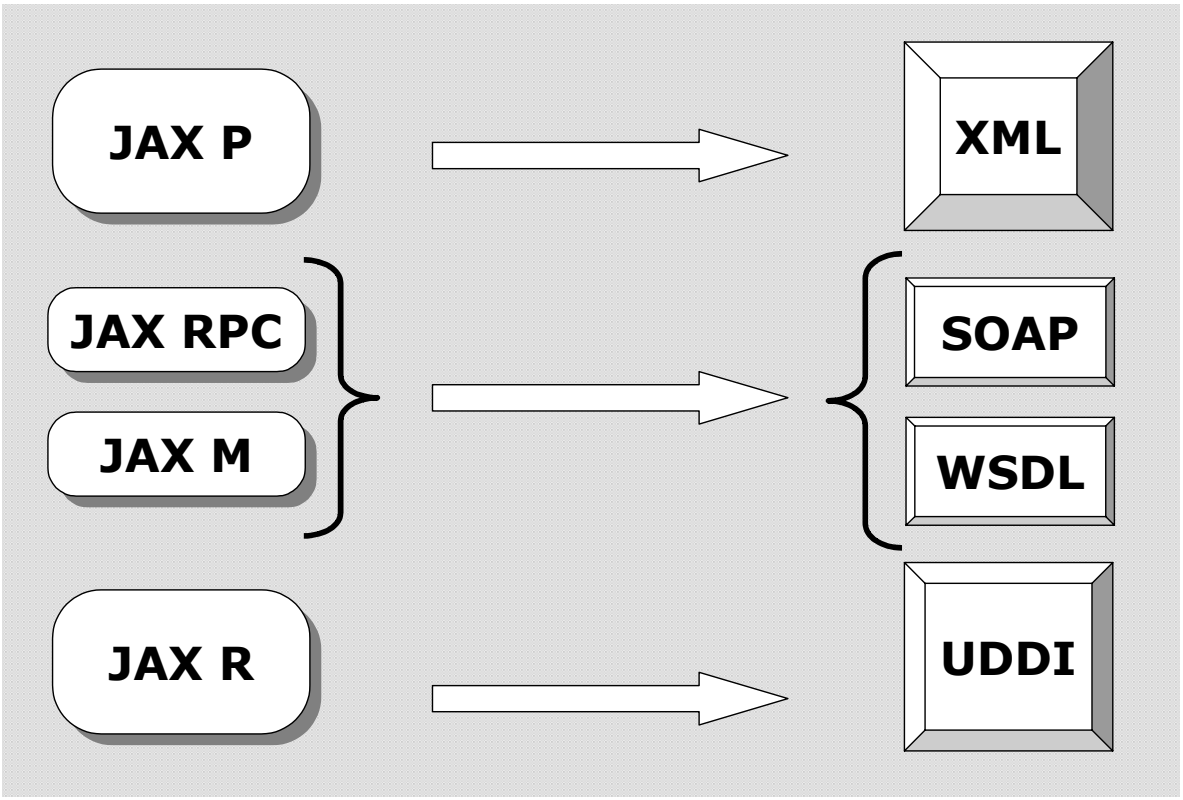


Fig. 4.6-1 equivalência entre as APIs Java e as tecnologias núcleo de Web Services.

Nosso estudo de caso prossegue com a descrição dos pontos relevantes de funcionamento do orçamento público e a definição de Web Services neste contexto de modo a publicar-se na Internet aspectos desta função do estado. Empregaremos a API JAX RPC na implementação de Web Services em Java baseado no modelo de comunicação síncrono funcionando sobre HTTP. O protótipo desenvolvido é composto pelo fornecedor de Web Services e o respectivo consumidor.

CAPÍTULO 5 – Divulgação do orçamento público via Web Services, em Java.

5.1 – Introdução

A estrutura do nosso estudo de caso baseia-se na divulgação de informações relativas a execução do Orçamento Público [62], de modo genérico em qualquer uma das esferas do Poder Executivo. Com este objetivo, orientamos o desenvolvimento de nosso estudo a fim de torná-lo válido nas quatro esferas do Poder Executivo : a União; os Estados; os Municípios e o Distrito Federal. Guardadas as proporcionalidades e os níveis de escalabilidade, o resultado a seguir apresentado poderá ser aplicado em qualquer das esferas citadas anteriormente.

Como objeto principal do estudo, temos a destacar o Orçamento Público no Brasil, o qual é regulamentado pela Lei No. 4.320, de 17/03/1964, que ao longo do tempo vem sofrendo adaptações e melhorias para que se torne adequada as necessidades sociais. No que se refere ao aparato legal, procuramos discorrer sobre os pontos considerados fundamentais para o objetivo do estudo e adotamos como regra padrão a omissão ao máximo nível, de citações referentes a leis, parágrafos, artigos e outros dispositivos legais, que nada tem a contribuir para o entendimento do que vem a ser o Orçamento Público de modo geral e no contexto delimitado para o presente estudo.

A seleção do tema Divulgação do Orçamento Público, através da tecnologia de Web Services, em Java, ocorreu como resultado de pesquisa realizada, sobre o comportamento da entidade pública, mediante a operação e a concretização dos objetivos descritos neste instrumento de trabalho. Concluímos que existe uma dinâmica, existe algo em movimento, ocorre uma transformação de planejamento, orçamento em ações e gastos de recursos públicos, que merecem uma análise mais aprofundada e mais detalhada. Sobre esta complexa dinâmica do Orçamento Público, escolhemos estudar de forma macro os três principais componentes quais sejam : o empenho, a liquidação e o pagamento. Para nosso estudo de caso, o pagamento é parte fundamental, pois encerra e finaliza toda a

dinâmica do orçamento. Porém, sem um breve esclarecimento sobre seus processos antecessores o mesmo ficaria desconexo e sem sentido no estudo de caso realizado.

O processo do estudo de caso nasce então no planejamento, passa para o orçamento, promessa, compromisso, intenção do governo de realizar investimentos em determinadas áreas como saúde, educação, segurança, etc. Neste ponto, inicia-se nosso estudo de caso, queremos empregar a tecnologia de Web Services para divulgar a qualquer tempo quanto o governo orçou (prometeu) investir nestas áreas. A parte final, é o pagamento, quanto foi realmente investido e gasto.

Nosso estudo de caso se propõe, de modo simplificado, em reunir estes dois dados e disponibilizá-los empregando a tecnologia de Web Services, que poderá ser acessado, neste estudo de caso, por qualquer cliente, dentro da rede privada do Estado (empregado aqui em seu sentido de Ciência Política como cita [52]) que tenha acesso a Internet e se proponha a implementar um cliente adequado. O trabalho se completa no próximo capítulo com a implementação concreta do Web Services e de um site cliente.

No que se refere a proposta do modelo de divulgação do orçamento público, o resultado esperado é discutido e apresentado em termos concretos. Oferecemos ainda pequena terminologia com o intuito de facilitar a compreensão e o entendimento do modelo e das partes que o compõe em sua realidade mas que tiveram de ser abstraídas em virtude do escopo adotado. Entretanto, temos como destaque no modelo os objetivos principais de utilização concreta da tecnologia de Web Services na Administração pública para a divulgação de informações e do emprego da linguagem Java para a integração destas informações. O tópico é concluído com o esquema ilustrativo da implementação funcional do modelo.

O cenário adotado constituiu-se de uma rede metropolitana municipal, mantida em perímetro seguro, conforme descrita em [58], composta por alguns dos principais órgãos. Descrevemos os aspectos relevantes de redes com esta similaridade e que podem ser aplicados às diversas esferas do Poder Executivo que contam com a mesma tipificação de parque instalado. Concluímos este tópico apresentando o esquema do cenário no qual se deu o estudo de caso.

Oferecemos tópico composto por elenco de justificativas para a aplicação da tecnologia de Web Services na Administração pública, nos quais procuramos demonstrar, a partir de temas selecionados, os principais fatos que a tornam adequada a este ambiente e concluímos com a proposição de duas possibilidades para a sua adoção.

O capítulo encerra-se com tópico que destaca a tecnologia Java e suas possíveis vantagens para a implementação de Web Services, considerando aspectos relevantes desta tecnologia em face ao estudo de caso apresentado e as características naturais de independência de plataforma e linguagem dos Web Services.

5.2 – Orçamento Público

Historicamente o orçamento no Brasil esteve a cargo dos governantes que dispunham do controle e utilização dos mesmos conforme seus próprios entendimentos e interesses. A padronização das atividades de contabilidade e finanças no setor público ocorrereu com a edição da Lei No. 4.320, de 17/03/1964, que instituiu o orçamento programa como sucessor ao orçamento tradicional ou normativo [56], ambos carentes de rigor, precisão técnica e padronização formal.

A instituição do orçamento programa, pela Lei No. 4.320/64, em vigor até a presente data, possibilitou ao governo em todas as esferas de poder : a União; os Estados; os Municípios e Distrito Federal, a formulação de programas, o acompanhamento e o controle da execução, a definição das responsabilidades e análise dos efeitos econômicos e sociais das ações implementadas [27] pelo emprego do recurso público.

5.2.1 – Plano Plurianual

Decorridos 13 anos, desde a Constituição de 1988, da incorporação do Plano Plurianual ao direito financeiro e orçamentário, observa-se, ainda, que, salvo algumas exceções, o mesmo tenha sido elaborado apenas como peça decorativa, para atender a exigência de cunho legal, sem compromisso com a definição de objetivos e metas, tanto para a administração pública quanto para a sociedade [56]. O Plano Plurianual, constitui-se peça fundamental para a elaboração do orçamento anual dos governos em todas as esferas de poder.

A inclusão do Plano Plurianual na constituição de 1988, tem por finalidade garantir o registro de metas para o estado e a sociedade nos quatro anos seguintes ao primeiro ano de mandato do atual governo. Assim, ao assumir, o governante atual, ainda encontra-se, executando orçamento vinculado ao último ano do Plano Plurianual do seu antecessor e deve planejar e apresentar o próximo Plano Plurianual que deve referir-se aos seus três anos de mandato restantes e ao primeiro ano do

próximo mandato. A partir do Plano Plurianual, nascem os orçamentos anuais que devem perseguir as metas econômicas, financeiras, sociais e de desenvolvimento descritas neste plano [56].

5.2.2 – Orçamento e Contabilidade

Para a implementação do orçamento, a Lei No. 4.320, de 17 de março de 1964, agrupou duas técnicas que a tradição vem utilizando em sistema de controle : o orçamento e a contabilidade [52]. No escopo deste trabalho, citamos partes necessárias ao entendimento do contexto, porém nosso foco central é demonstrar aspectos relevantes do orçamento que nosso estudo de caso empregará na proposição de utilização de Web Services.

O orçamento evoluiu para aliar-se ao planejamento, surgindo o orçamento-programa como especialização [52]. Existe uma corrente de pensamento que considera o orçamento ligado intrinsecamente ao planejamento, colocando o mesmo em nível de igualdade com a contabilidade e o próprio planejamento e jamais como subproduto ou em nível inferior a qualquer um destes [52].

Na prática, o orçamento deve funcionar como ponte, peça de ligação entre os sistemas de planejamento e finanças [52], tendo na contabilidade o apoio para registro e armazenamento de todas as atividades inerentes a administração dos recursos públicos que fluem entre planejamento, orçamento e finanças.

A contabilidade, parte integrante e importante para o registro formal das atividades orçamentárias e financeiras, constitui-se, modernamente em processo gerador de informações sobre os atos realizados pela administração pública em termos financeiros [52]. Permitindo o acompanhamento e o desenrolar das atividades concretizadas e registradas. Tornando-se assim, fonte importante para a análise de dados e produção de informações como citado em [56].

Precisamos porém diferenciar as duas técnicas : de orçamento e contabilidade. Enquanto a contabilidade constitui-se o instrumento que possibilita a informação para tomada de decisões, controle e avaliação de desempenho [52] após o mesmo ter ocorrido e ter sido concluído. O orçamento deverá refletir informações políticas e programas para possibilitar o controle gerencial, aliadas a um sistema de quantificação física para a mensuração das ações governamentais [52].

5.2.3 – Instrumento na dinâmica da administração

Colocando-se o orçamento como sistema dinâmico funcionalmente semelhante a um duto ou a uma ponte de ligação entre os sistemas de planejamento e finanças, torna-se possível a operacionalização dos planos, porque monetariza, isto é, coloca-os em função dos recursos financeiros disponíveis, permitindo que o planejador tenha o controle preciso, em face das reais disponibilidades dos recursos financeiros [52] que deve destinar para cada ação ou projeto de acordo com prioridades preestabelecidas.

Assim, o orçamento apresenta-se fundamentalmente como um instrumento de que o administrador dispõe para equacionar o futuro em termos realísticos [28], como um curso de ação, um programa operacional [52]. Está definição apresenta o caráter dinâmico do orçamento público que se deseja capturar e divulgar, em uma pequena parcela, através da aplicação de Web Services e suas facilidades tecnológicas neste estudo de caso.

Destinado a atender a dinâmica da sociedade [29], dentro de linhas gerais planejadas para um certo período de tempo, um ano, no caso dos orçamentos públicos brasileiros. A dinâmica do orçamento não pode prender-se a seguir de modo rígido e rigoroso este planejamento anual para o emprego de recursos, se assim o fosse teríamos o orçamento monolítico anual.

A integração planejamento/orçamento constitui-se a tônica atual, servindo como ferramenta capaz de concertar as distorções administrativas e remover impecilhos institucionais que dificultam a modernização dos métodos e processos administrativos no Brasil [52].

5.2.4 – Ponte de ligação entre planejamento e finanças

O orçamento, portanto, é uma técnica cujo maior significado moderno consiste precisamente em ligar os sistemas de planejamento e finanças através da expressão quantitativa financeira e física de programas de trabalho do Governo, valendo este conceito também para o orçamento empresarial [52].

Por estar situado entre os sistemas de planejamento e finanças, o sistema de orçamento constitui-se lugar adequado para o levantamento e a elaboração de estatísticas de controle para apoiar a tomada de decisão [52].

Modernamente o orçamento constitui-se bem mais que uma consolidação entre planos físicos e recursos públicos aplicados em ações de governo das mais variadas naturezas : é um instrumento de trabalho [52] de comportamento dinâmico. Neste sentido, é possível empregar o orçamento como meio de descentralização administrativa, de delegação de competência e de apuração de responsabilidades, não só da organização, mas também dos gestores, de modo que a aprovação do orçamento pelo poder legislativo, signifique a autorização para a ação e, concomitantemente, o início do processo de controle [52].

5.2.5 – Lei que dá início a execução orçamentária

A Lei do Orçamento anual disporá critérios e formas de liberação por parte do Poder Executivo dos recursos destinados aos Poderes Legislativo, Judiciário e Ministério Público, conforme disposto na Constituição Federal [52].

Com aprovação da Lei do Orçamento anual, teríamos o início do então orçamento geral do Governo, não importando a esfera em que situe, expressão macro da posição das finanças governamentais e, para cada projeto e atividade, a expressão micro, base e autorização para a ação administrativa dos respectivos responsáveis [52]. Daí porque entendermos que o orçamento constitui-se também instrumento de controle gerencial [27], por possibilitar informações para comparações e avaliações de caráter gerencial [28], tais como as da economicidade, da eficiência, da eficácia e da efetividade [52].

Temos o caráter distribuído do orçamento, que contribui para que se expanda na esfera administrativa do governo, a necessidade de tecnologias capazes de trocar informações em ambientes distribuídos e heterogêneos. A concepção moderna do orçamento em base gerencial traduz os órgãos como centros de responsabilidades ou de resultados e também de decisão com certo horizonte de alcance limitado, onde se identificam as responsabilidades decisórias dos respectivos gestores pela utilização dos recursos que lhes são confiados para possibilitar a execução das ações nas áreas de responsabilidade em que o Estado desempenha suas atribuições [52]. Evidentemente estas responsabilidades e ações são avaliadas mediante o emprego de indicadores de qualidade adequados às áreas em que atuam tais gestores [52].

5.2.6 – Formalização e padronização

Como é sabido a Lei 4.320 de 17/03/1964, abrange a União, os Estados, os Municípios, e o Distrito Federal, possibilitando, assim a existência de normas homogêneas para todo o País e facilitando o levantamento de dados estatísticos financeiros e na realização dos programas de trabalho, bem como a consolidação dos orçamentos e balanços do setor público brasileiro [52]. Esta uniformidade imposta pela lei para todas as esferas, possibilita ao nosso presente estudo de caso a aplicação em qualquer das esferas mencionadas, com o mesmo resultado de validação, para o emprego de Web Services, preservando-se, obviamente, particularidades de escala e complexidade proporcionais ao aumento da capilaridade de partes integrantes do cenário a ser considerado, no que diz respeito a divulgação de partes relevantes do orçamento público.

Dispositivos da Constituição no âmbito da legislação concorrente, a competência da União limitar-se-á a estabelecer normas gerais, que não excluem a competência suplementar dos Estados [52].

Os municípios, também poderão legislar de forma suplementar, de acordo com a Constituição do Brasil, no que couber, à legislação federal ou estadual. Neste caso, o Município pode estabelecer para si normas específicas de controle interno e de administração financeira [52] incluindo mecanismos de troca de informação entre seus entes e formas de divulgação de suas atividades.

O princípio da entidade fundamenta-se no fato de que o orçamento e a contabilidade constituem-se instrumentos que possibilitam informações sobre transações/operações das entidades jurídicas governamentais, tais como expressas na letra da lei, inclusive das suas fundações, autarquias, empresas públicas e sociedades de economia mista, as quais produzem reflexos sobre os respectivos patrimônios [52] e contribuem de forma concreta para a transformação dos recursos planejados e orçados transformarem-se em ações de governo.

Entretanto a observação sobre o princípio da entidade, possui importância fundamental, pois o Estado (em sua acepção ampla) constitui-se também um ente econômico, além de social, daí porque os orçamentos de todas as organizações governamentais serem apresentados, agregados e aprovados em uma só lei, conforme disposto na Constituição Federal, que lança as bases e normas gerais para a padronização dos orçamentos e dos balanços consolidados de forma nacional [52].

5.2.7 – Lei de Responsabilidade Fiscal e o Orçamento

A Lei de Responsabilidade Fiscal(lei complementar num. 101/2000, de 4-5-2000), passou a reger com maior rigor e exatidão certos procedimentos da Lei 4.320 de 17/03/1964 [52]. Como sua denominação indica, dispõe sobre normas de comportamento ético para o gestor público no trato do patrimônio público que lhe fora confiado. Está lei dispõe sobre normas a serem seguidas nos procedimentos para com as finanças públicas [52]. E vai além, posto que abraça procedimentos de natureza técnico-econômica que envolvem análise do custo-benefício, análise do fluxo de caixa, contabilidade financeira melhor planejada, contabilidade gerencial para possibilitar informações para avaliação de desempenho sob enfoques da eficiência, da economicidade, da eficácia e da efetividade [52]. Todos os gestores, de todas as unidades que sejam responsáveis pelo trato com os dinheiros e /ou o patrimônio público, em todas as esferas de poder estão sujeitos as exigências da Lei de Responsabilidade Fiscal [52].

A repartição da previsão dos recursos para a composição do Tesouro do Estado, entre suas entidades, para a execução de programas e ações de governo, que serão concretizadas em benefícios sociais, constitui-se, de forma simplificada, o orçamento público [52].

5.2.8 – Execução do orçamento

Trata-se a execução do orçamento de disciplinar a utilização dos recursos das unidades orçamentárias e/ou administrativas, com o objetivo de fazer com que as ações governamentais sejam executadas sem solução de continuidade e assim cumprir a missão com eficiência, buscando a eficácia, nos resultados esperados e ainda evitando desperdícios e com isso protegendo o patrimônio público [52].

A Lei 4.320 de 17/03/1964, através artigo designado para este fim, introduziu o sistema de programação da execução orçamentária. Tem-se a impressão de que programar é uma tarefa assim como datilografar um texto. Mas não. A programação é um processo contínuo em Administração [52].

Aprovado o orçamento, isto é, aprovado o plano de trabalho do governo e os respectivos limites financeiros para sua execução, obedecendo sempre a estimativa e dentro do esquema de recursos que o Governo é autorizado a arrecadar, começa a tarefa de tornar operante o orçamento [52], inicia-se a execução, a parte da transformação dos planos, do planejamento, que foi expresso em bases orçamentárias, estimativas de custo financeiro e gasto público em ações concretas que produzirão benefícios para a sociedade.

Ao fixar com a aprovação do orçamento, os limites dentro dos quais o executivo deve trabalhar, cumpre estabelecer o quadro de cotas trimestrais, para cada unidade orçamentária [52]. Tem-se neste caso não só um sistema de programação da despesa, mas também um instrumento mediante o qual se busca agilizar as ações governamentais através da descentralização, não apenas orçamentária, mas também administrativa, porque, ao aprovar as cotas, o Executivo estará delegando às chefias das unidades respectivas responsabilidades (descentralização do processo decisorial) pela movimentação dos créditos orçamentários, juntamente com os programas que estas unidades distribuídas devem executar [52]. Pode-se perceber que a própria lei que rege a administração do orçamento estimula e incentiva o modelo de administração descentralizado e distribuído, imponto ao gestor público setorial ações sobre a parte do orçamento que lhe é cabida, o que nos remete a modelos de computação distribuída capazes de atender a demanda do modelo de negócio caracterizado. Entre estes, Web Services constituem-se uma das propostas mais adequadas considerando-se a realidade brasileira atual.

Como uma das propostas encontradas na literatura [52] , o cálculo para a determinação de cotas poderá ser realizado considerando, como oportunidade, o total das despesas fixas e das que devam ser realizadas em função das variações impostas pelo cronograma de obras e outros fatos que não se conformam com o decurso linear do tempo.

Implantado e regido pelo o estabelecimento de cotas, a unidade orçamentária saberá que quantia poderá empenhar por trimestre ou bimestre, ou mês a mês (a decisão desta periodicidade fica a cargo da esfera de poder), em coordenação com o serviço de pessoal e com o setor de planejamento da entidade [52].

O planejamento setorial deve ser extremamente estimulado no país a fim de que possam ser atendidos com maior precisão os anseios e necessidades dos cidadãos, estando este planejamento mais próximo, inclusive fisicamente, poderá faze-lo com maior rapidez e agilidade [52].

Conforme observa a programação orçamentária, bem como programação financeira de desembolso, ambos constituem-se instrumentos de real importância para conferir disciplina concreta na utilização dos recursos públicos que devem ser apresentados em atendimento a decreto lei respectivo que os impõe a Governos Estaduais e Municípios como norma geral a ser atendida [52].

Para estabelecer o funcionamento contínuo do orçamento a Constituição Federal determina que as entregas dos recursos correspondentes às dotações orçamentárias pelo Executivo aos

demais poderes se darão até o dia vinte de cada mês. Isto significa que esses Poderes deverão organizar as respectivas programações , tal qual dispõe o presente artigo, que ora se comenta [52].

De acordo com o estabelecido em lei, até trinta dias após a publicação do orçamento nos termos que dispuser a Lei de Diretrizes Orçamentárias (LDO), o Poder Executivo estabelecerá a programação financeira e o cronograma de desembolso, conforme o disposto na Lei de Responsabilidade Fiscal (LRF) [52]. Temos então dois instrumentos de controle impostos por esta lei : a programação financeira e o cronograma de desembolso. A programação financeira é a mesma que a programação de despesa, na qual a distribuição das dotações para as despesas pelos setores da administração deve ser realizada bimestralmente e considerar sempre a produção de cada um deles, a fim de que os recursos ou insumos sejam utilizados com o máximo de eficiência, cujo o objetivo, mediante utilização disciplinada desses recursos, é evitar o desperdício e assim possibilitar a concretização das ações previamente planejadas [52].

O cronograma de desembolso constitui-se em um instrumento de controle de Tesouraria ou de Caixa, em que se prevêem as receitas e os pagamentos das obrigações, que vão sendo assumidas à medida que o orçamento vai sendo executado. O cronograma de desembolso deve ser elaborado, para melhor controle financeiro, pelas espécies de caixas subordinados ao controle da Tesouraria [52].

Existe entretanto na LRF dispositivo de segurança que previne o endividamento e o crescimento do déficit orçamentário, o qual determina, em se verificando ao final de um bimestre, que a realização da receita não comporta o cumprimento das metas de Resultado Primário ou nominal estabelecida no anexo de Metas Fiscais, os Poderes e o Ministério Público promoverão, por ato próprio, e nos montantes necessários, nos trinta dias subsequentes, limitação de empenhos e movimentação financeira, de acordo com os critérios estabelecidos na LDO [52] Com este dispositivo, havendo queda na arrecadação, ficam suspensos e adiados as programações anteriores descritas nos cronogramas de desembolso e programação financeira.

A definição do sistema de cotas tem por finalidade permitir regulação e controle dos gastos públicos, evitando assim entre outros o endividamento. Assim temos atrelado ao sistema de cotas o problema da estimativa da receita, que deve ser feita com critério e objetividade [52]. Como se pode perceber o sistema de cotas visa também que a entidade mantenha comportamento regular na utilização de seus numerários. Com isso evitar-se-ão os déficits de tesouraria setorial, que obrigam a entidade a recorrer a operações de créditos [52] as quais são indesejáveis.

Aqui exerce grande influência positiva para a perfeita harmonia dos planos e objetivos estabelecidos na execução orçamentária, a adoção de um calendário de arrecadação de impostos e outros tributos, de modo que o grosso da receita possa ser recebido na época em que é maior o peso das responsabilidades de tesouraria [52] considerando-se todos os níveis de execução do orçamento, desde a unidade central até as unidades orçamentárias periféricas.

Incluem-se entre as despesas das unidades orçamentárias, além das despesas explicitadas no orçamento outras despesas, operações de natureza financeira e extra-orçamentárias. O aperfeiçoamento da lei trata em um de seus artigos de disciplinar os fluxos de caixa e não só da execução orçamentária, mas também de outras operações. Esta disciplina deve ser conjugada com a programação das despesas ou programação financeira. Desta forma recai, inevitavelmente, que a responsabilidade de elaborar a distribuição de cotas é do órgão fazendário da entidade governamental em colaboração com as demais unidades [52]. Os responsáveis pela programação da cotas trimestrais terão sempre em vista:

- as disponibilidades existentes;
- a afluência provável da receita própria;
- a afluência provável da receita transferida;
- orçamento aprovado com discriminação por unidade orçamentária;
- outros débitos ou compromissos, como os oriundos de créditos especiais, abertos nos últimos quatro meses do exercício recém-encerrado, os restos a pagar, os compromissos da dívida pública e outros dessa natureza;
- outros compromissos e outras circunstâncias que possam influir no comportamento efetivo das receitas e na realização da despesa [52].

Do exposto até o momento, pode-se inferir que as cotas trimestrais não constituem-se de pequenos orçamentos que só podem ser alterados por lei, pelo contrário, permitem flexibilidade, de modo que se possa atender às circunstâncias da conjuntura econômica [52]. Permitindo ao gestor público, de todos os níveis, operar a característica dinâmica da execução orçamentária distribuída entre as várias unidades, de acordo com contingências que podem apresentar-se no campo das finanças, da economia, oriundas de causas naturais e mais freqüentemente por demanda política.

Entretanto a Lei No. 4.320, permite uma série de operações de caráter orçamentário as quais permitem, ao longo do exercício, a alteração do orçamento inicial, para cada unidade gestora [52].

Entre estas operações podemos citar a suplementação, aumento do orçamento de determinada ação, a redução, como o nome diz a diminuição do orçamento para determinada ação, créditos especiais e créditos extraordinários [52]. De modo que registra-se no início do período a coluna que chama-se dotação inicial e está poderá sofrer acréscimos e decréscimos de acordo com os atos da Administração [52]. Essas operações foram aqui citadas somente a título de informação, pois não serão consideradas dentro do escopo de aplicação do estudo de caso do qual este trabalho tem como objetivo.

Na realidade, o que a lei pretendeu foi criar um instrumento sem aquela rigidez característica das contas duodecimais e colocar à disposição do processo financeiro um instrumento flexível, que permita executar o orçamento em função das circunstâncias reais em que a entidade operar [52].

5.2.9 – Aspecto dinâmico da despesa orçamentária

A realização concreta do orçamento para este estudo de caso, constitui-se em transformar o planejamento que deu origem em orçamento em despesa concreta. Este considera-se o aspecto dinâmico do orçamento, o qual é composto por diversos processos, que juntos produzem o resultado final esperado ou seja o pagamento da despesa. A despesa é a aplicação de recursos mediante a qual qualquer organização, independente de sua natureza jurídica, procura alcançar seus objetivos e, consequentemente cumprir com a sua missão, não interessando que tenha tempo de existência definido (temporária) ou indefinido. Interessa que seja importante para o funcionamento da organização [52].

Na administração governamental, ainda que não seja diferente essa premissa, qualquer que seja a despesa, independentemente de seu objeto, só poderá ser realizada quando a lei autorizá-la [52].

Em realidade, o que a lei autoriza é a criação daquilo que é a razão da despesa, ou seja, a ação que será empreendida para a consecução dos objetivos da entidade [52].

Autorizada, a despesa governamental deverá obedecer certas regras que lhes são impostas para sua execução tais como : a programação; a requisição pelo órgão interessado; a autorização por aquele que é responsável pela decisão; a licitação e posteriormente; o seu empenho, o qual na sua realização deverá atender às seguintes condições :

- deve emanar da autoridade competente : o Chefe do Executivo, em princípio, e, por delegação de competência, o Diretor ou Secretário de Fazenda, os Diretores dos demais Departamentos, ou outro funcionário devidamente credenciado. Em suma, estes são os ordenadores de despesa;
- cria para o estado obrigação de pagamento;
- pode esta obrigação de pagamento ser pendente ou não de implemento de condição [52].

5.2.10 – Empenho, ligação entre orçamento e execução orçamentária

Constitui-se o empenho em parte fundamental em nosso estudo de caso entre o orçamento e a execução orçamentária. Na verdade o empenho constitui-se em uma das fases mais importantes por que passa a despesa pública, obedecendo a um processo que vai até o pagamento [52].

O empenho não cria obrigação e, sim ratifica a garantia de pagamento assegurada na relação contratual existente entre o Estado e seus fornecedores e prestadores de serviços [52].

Administrativamente poderíamos definir o empenho da seguinte forma : ato de autoridade competente que determina a dedução do valor da despesa a ser executada da dotação consignada no orçamento para atender a essa despesa. É uma reserva que se faz, ou garantia que se dá ao fornecedor ou prestador de serviços, com base em autorização e dedução da dotação respectiva, de que o fornecimento ou serviço contratado será pago, desde que observadas as cláusulas contratuais. Todavia, não é só dos contratos, convênios, acordos ou ajustes que resultam obrigações do Estado [52]. Outra forma de definir empenho é oferecida pela comissão de reforma da Lei 4.320/64, reunida em meados de 1996, a qual discutiu o assunto exaustivamente e elaborou definição sob o ângulo de destaque da dotação. Seria então o caso de definir empenho como :

"Empenho de despesa é o ato emanado de autoridade competente que vincula dotação de créditos orçamentários para pagamento de obrigação decorrente de lei, contrato, acordo ou ajuste, obedecidas as condições estabelecidas." [52]

Estamos tratando neste tópico e em todo este capítulo a palavra Estado sempre em seu sentido amplo, em uma conotação de Ciência Política, significando Poder Público, e não no restrito de Estado membro da Federação [52]. Isto torna-se válido inclusive para nosso estudo de caso que aborda, neste capítulo, a sua aplicabilidade a qualquer uma das esferas do poder executivo regida pela Lei 4.320/64.

Para que se possa empenhar, existe a necessidade de disponibilidade anterior ao empenho de créditos autorizados. A lei possui dispositivos para vedar empenhos que excedam este limite. Poderão ser feitos, entretanto, tantos quantos empenhos forem necessários, mas seu somatório não poderá ultrapassar o montante da dotação [52].

O empenho é o instrumento de que se serve a Administração a fim de controlar a execução do orçamento. È através dele que o Legislativo se certifica de que os créditos concedidos ao Executivo através do orçamento aprovado, estão sendo obedecidos [52].

Constitui-se o empenho em instrumento de programação, pois, ao utilizá-lo racionalmente, o Executivo tem sempre o panorama dos compromissos assumidos e das dotações ainda disponíveis. Isto constitui uma garantia para os fornecedores, prestadores de serviços e empreiteiros, contratantes em geral [52] e demais entidades da sociedade que mantenham relacionamento econômico financeiro com o Executivo.

O conceito de empenho, pressupõe anterioridade. O empenho é *ex-ante*. Daí o receio de haver uma definição legal de empenho meramente formal. Isto decorre, no entanto, da prática brasileira que é a do empenho *ex-post*, isto é, depois de executada a despesa, apenas para satisfazer o dispositivo legal, ao qual o Executivo não quer obedecer, por falta de capacidade de programação [52].

Pelo conceito da Lei 4.320/64, não há empenho posteriori. O grande problema, entretanto, está contido na expressão da lei " ... realização de despesa ... " que por muito tempo foi registrada com o significado exclusivo de pagamento. Em realidade a expressão em si tem outro significado, ou seja, nenhuma compra de bens ou serviços, ainda que de utilização futura, ou assunção de encargos sociais e financeiros, será efetivada (realizada) sem prévio empenho ou provisão orçamentária [52].

Torna-se importante esclarecer que o documento Nota de Empenho é simplesmente um mecanismo utilizado pelo Poder Público para informar sobre a materialização da garantia de pagamento assegurada pela relação contratual entre o Estado e terceiros, ou ainda para cumprimento de obrigações de pagamentos oriundos de mandamentos constitucionais e de leis ordinárias [52].

5.2.11 – Tipificação do empenho

Quanto a tipificação de empenhos, a lei flexibiliza a existência de três categorias, duas de forma explícita e uma de forma implícita. Nas categorias de forma explícita temos os empenhos por estimativa e prévio. Na forma implícita temos os empenhos chamados corriqueiramente de empenhos ordinários que tem como característica o conhecimento prévio de valor exato e das datas em que deverá iniciar e encerrar o seu objeto.

A lei dispõe que será feito por estimativa o empenho de despesa cujo montante não se possa determinar [52]. Emprega-se esta categoria de empenho se não se sabe, ou não se pode calcular o montante exato da despesa, faz-se o empenho sempre prévio e por estimativa, o valor exato da despesa poderá ser conhecido no exercício de origem ou no exercício subsequente [52]. Podem ser empenhadas por estimativa despesas cujo valor exato seja de difícil identificação e aquelas que obrigatoriamente são realizadas, dada a sua importância ou natureza [52].

Com a finalidade de flexibilizar os procedimentos de gestão do orçamento, a lei dispõe de dispositivo que possibilita outra faculdade ou exceção aos empenhos, que permite adotar sempre a regra do empenho prévio [52]. Podendo o empenho ser feito pelo total do objeto contratado para pagamento em parcelas fixas. Se exceder o limite do exercício financeiro, a parte não liquidada ou não paga figurará em Restos a Pagar não Processados. Nesta modalidade de empenhos podemos incluir as despesas com pessoal, sobretudo vencimentos e vantagens diretas, com exceção de diárias, ajudas de custo e outras de caráter específico, bem como os contratos de financiamento [52].

A Nota de Empenho constitui-se no documento utilizado para registros de operações que envolvem despesas orçamentárias realizadas pela Administração pública, incluindo órgãos de Administração indireta [52]. Todos os entes para os quais tenham sido alocados recursos orçamentários serão obrigados a adotar a Nota de Empenho para movimentação destes recursos.

Elementos obrigatórios que devem compor a estrutura do empenho, além destes , se poderá incluir outros elementos que se julgar necessários para aperfeiçoar o controle interno :

- nome do credor, se possível endereço completo do credor;
- especificação da despesa, isto é a classificação da despesa segundo o plano de contas;
- importância da despesa, em algarismos e por extenso;

- dedução da despesa do saldo da dotação orçamentária própria, a declaração de que a importância empenhada foi abatida do saldo da dotação pela qual se fez o empenho, neste ponto é que se concretiza a garantia do credor[52].

Em [52] sugere-se que a declaração de liquidação poderá ser feita na própria Nota de Empenho, o que elimina trâmite burocrático e uso indevido de mais papel. Porém, essa sugestão dificilmente é implementada, devido a grande insegurança, no que se refere ao domínio e ao conhecimento profundo dos processos relacionados, como pudemos observar em nossa pesquisa de campo junto a usuários deste tipo de sistema.

Nos casos em que as despesas resultem de contratos, acordo e convênios, são obrigatórias a emissão da Nota de Empenho e a sua entrega ao contratante do Estado, a fim de que o mesmo tome conhecimento da reserva feita em seu favor, que lhe será paga, observadas as condições impostas por está lei. Em casos especiais, através da Justiça, serve de comprovação desse mesmo crédito [52].

A emissão da Nota de Empenho será feita em quantas vias forem necessárias, bastando entregar ao contratante (fornecedor, empreiteiro de obras ou prestador de serviços), a primeira via do documento, a fim de juntá-la aos demais documentos para instruir o processo de liquidação, de que trata a lei [52].

Como afirmamos, empenho significa anterioridade e, a fim de que esta fique comprovada, os documentos comerciais devem fazer referência ao número da Nota de Empenho entregue ao contratante.

5.2.12 – Mecanismo de liquidação do empenho

Vários mecanismos foram criados pela lei e devem funcionar de maneira integrada, hierárquica e principalmente sincronizada para controlar e gerir o fluxo dos recursos públicos, que tem origem no orçamento, são comprometidos ou alocados por empenhos e passam pela fase de liquidação, mecanismo necessário de conferência, aferição e verificação duplicada para controle.

A presença da fase de liquidação tornou-se absolutamente necessária para contrabalançar a dada extensão atribuída ao conceito de empenho que materializa-se pela emissão do documento Nota de Empenho. A liquidação da despesa constitui-se o dispositivo que permite à

Administração reconhecer a dívida como líquida e certa a obrigação de pagamento, desde que as cláusulas contratadas tenham sido rigorosamente cumpridas [52].

Vale acentuar que o empenho não ocorre no pagamento, mas antes, na autorização. O pagamento constituísse a segunda fase da despesa. A observação é pertinente porque algumas administrações incorrem neste engano [52].

Como definição de liquidação oferecemos os temos da lei : liquidação da despesa consiste na verificação do direito adquirido pelo credor, tendo por base os títulos e documentos comprobatórios do respectivo crédito [52]. A despesa passa, entre outras, pelas seguintes fases : o empenho, a liquidação e o pagamento [52], este transcorrer de fases constitui, em nossa observação para fins de estudo de caso, em potencial nicho para plicações de Web Services no aspecto dinâmico do orçamento, que possuindo este caráter, pode ter como aliada esta tecnologia para, inicialmente, divulga-lo, permitindo que aplicações do próprio Estado possam, em uma fase inicial, acompanhar sua dinâmica a partir de suas próprias plataformas e em níveis de granularidade de informações diferentes.

A liquidação é, pois, a verificação do implemento de condição. Trata-se de verificar o direito do credor ao pagamento, isto e, verificar se o implemento de condição foi cumprido. Isto se faz com base em títulos e documentos. Porém há um ponto central a considerar : é a verificação objetiva do cumprimento contratual. O documento constitui-se apenas o aspecto formal da processualística [52]. A fase da liquidação deve comportar a verificação in loco do cumprimento da obrigação por parte do contratante [52].

5.2.13 – Pagamento da liquidação

Após a fase de liquidação, segue a de pagamento, definida como : a ordem de pagamento é o despacho exarado por autoridade competente, determinando que a despesa seja paga. A ordem de pagamento é exatamente a última fase do estágio da despesa, de que trata a lei [52]. Ela deverá ser exarada no processo da despesa pela pessoa legalmente investida na autoridade de ordenar pagamentos. Entretanto essa autoridade poderá ser delegada [52]. A delegação de autoridade para ordenar pagamento constitui-se em questão de controle interno cabendo a cada entidade optar pela delegação que melhor se adapte ao seu trâmite funcional, dentro das normas da lei.

Para a formalização dos pagamentos e sua estruturação, o problema com que aqui nos deparamos é muito mais de conceituação, organização e comportamento administrativo e deve ser

encaminhado da melhor forma possível in loco [52]. Pois as rotinas de pagamento devem submeter-se aos dispositivos tecnológicos mais modernos que estiverem ao alcance da Administração como o sistema bancário e suas amplas e seguras facilidades, ficando para o ente administrativo somente a tarefa de comandar e controlar o financeiro disponível.

A lei obriga, entretanto, que a entidade deverá possuir, de forma institucionalizada, um setor para realizar os pagamentos, geralmente denominado de Tesouraria. Pode constituir-se em uma caixa única ou funcionar descentralizadamente por caixas recebedoras e caixas pagadoras. Para os casos de cidades grandes as tesourarias podem ainda ser regionalizadas ou distribuídas por bairros ou regiões administrativas [52].

O pagamento por intermédio da rede bancária pode ser feito por meio de cheque nominal, prática já em desuso, ou por depósito direto na conta do credor, do modo como já se faz na maioria dos Governos para o pagamento de pessoal. A entidade poderá também manter saldo suficiente em banco e autorizar o credor a colocar em cobrança sua duplicata, a qual será compensada como se fora um cheque [52].

A modalidade a escolher, portanto, fica a critério da Administração. A mais simples é a utilização da rede bancária para os recebimentos de pagamentos efetuados pela Administração. Entretanto, esta modalidade não exclui a necessidade legal e institucional da existência de um órgão de Tesouraria , inclusive para comandar os procedimentos e controlar os saldos individualizadamente por banco e por conta [52].

Estes são os aspectos fundamentais sobre a fase de pagamento que estão no escopo de nosso estudo de caso. Outras questões de funcionamento e modalidades de pagamentos são tratadas pela lei que não dizem respeito ao objetivo pretendido.

5.2.14 – Estrutura de funcionamento simplificado

Abstraindo a grande maioria dos detalhes, descreveremos o funcionamento simplificado da execução orçamentária baseando-nos nos aspectos principais e considerando um caso trivial genérico de um órgão que empenha uma ação de governo, devendo a mesma ser liquidada e finalmente paga. Toda esta narrativa se dará sempre focando o objetivo simples de esclarecer o funcionamento do mecanismo dinâmico da execução orçamentária, objetivo deste estudo :

- aprova-se o orçamento contendo recursos destinados a cada órgão para que estes dêem prosseguimento as ações de governo esperadas pela população;
- empenham-se as importâncias, previamente orçadas e disponibilizadas, sempre obedecendo os limites das cotas trimestrais, para que as ações sejam realizadas pelos órgãos da administração em suas diversas áreas de atuação
- emiti-se então o documento Nota de Empenho, que será chamado de empenho, apenas;
- realiza-se a ação descrita no empenho;
- ser liquida-se o empenho, após a ação realizada;
- produz-se o pagamento da ação que foi realizada, após a liquidação.

Para maior facilidade e entendimento, oferecemos a Figura 5.2.14-1 funcionamento simplificado da execução orçamentária, que possui numeração dos passos. A princípio tal descrição pode parecer trivial, porém devemos levar em conta alguns fatores que a tornam um potencial sistema computacional distribuído altamente complexo, vejamos alguns pontos : a escalabilidade no número de órgãos em um orçamento, do número de ações programadas por órgãos, do número de empenhos, do número de liquidações e de pagamentos. Junte-se a isto a distribuição geográfica destas ações ocorrendo em um cenário de vários municípios sob o mesmo estado ou ainda, em escala menor, vários órgãos espalhados por vários bairros em grandes cidades. O funcionamento simplificado apresentado na ilustração aplica-se perfeitamente a escalabilidade ampliada como exemplo, demonstrando o emprego da ilustração trivial como cenário de ensaio válido.

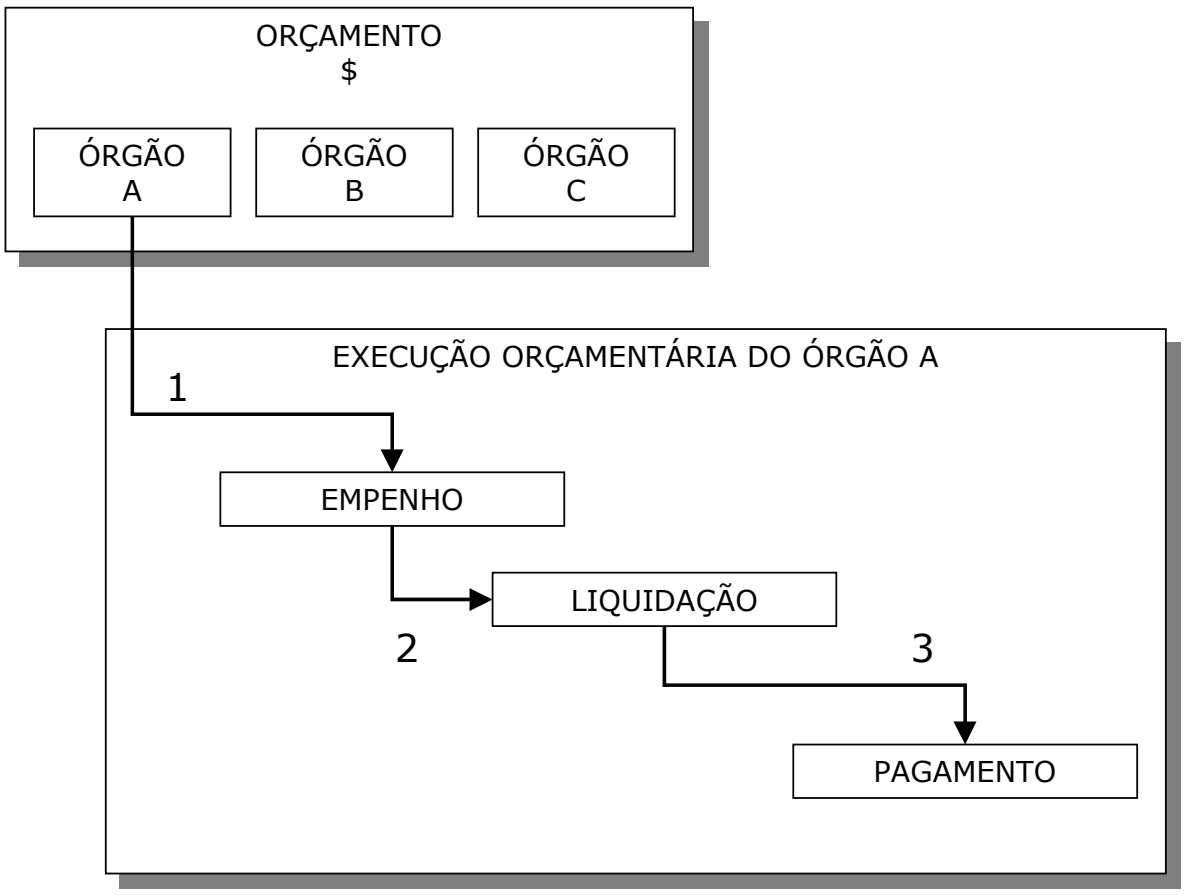


Fig. 5.2.14-1 funcionamento simplificado da execução orçamentária.

5.3 – Divulgação do Orçamento Público

O presente estudo de caso tem como foco a divulgação de aspectos dinâmicos do orçamento público, de modo a demonstrar, como a tecnologia de Web Services pode ser aplicada no contexto da Administração pública.

Tendo o poder executivo a tarefa de gerir e ordenar grande parte do recuso públicos, elaborando a lei de orçamento e apresentando para sua aprovação, como citado em [52], de modo unificado com os demais poderes Legislativo e Judiciário, nosso estudo de caso tem como base central o orçamento gerido pelo Poder Executivo, sem entretanto considerar suas relações orçamentárias com os demais poderes.

A escolha do termo divulgação, teve por objetivo enviar a pesquisa em direção de exemplo simplificado de aplicação da tecnologia, a fim de que o objetivo maior não viesse a sofrer qualquer tipo de interferência ou obstáculo para o seu mais perfeito entendimento que, apesar de

formalmente definido no início deste trabalho, também pode ser resumido de forma simplificada em uma única frase : empregar a Web Services, em Java, para tornar as operações realizadas no orçamento público acessáveis, através da Internet, por uma ampla quantidade de clientes em potencial.

Assim, o termo divulgação do orçamento público, refere-se tão somente ao exemplo, ao estudo de caso em que empregamos a tecnologia de Web Services, em Java, para demonstrar de forma concreta como a mesma pode tornar-se uma proposta de solução, em computação distribuída, no âmbito da administração pública do orçamento.

5.3.1 – Terminologia

A linguagem empregada pela literatura específica da área de Administração pública consultada, como em [52], [56] e [162], compõem-se de termos próprios com significados amplos e pouco familiares. A seguir oferecemos os principais termos e definições que serão empregados deste ponto em diante, para a exposição do estudo de caso realizado :

- orçamento público – lei de iniciativa do Poder Executivo que estima a receita e fixa a despesa da administração pública. É elaborada no exercício corrente (ano fiscal) para depois ser aprovada pelo Poder Legislativo, e vigorar no exercício seguinte;
- orçamento programa – originalmente , Sistema de Planejamento, Programação e Orçamentação, introduzido nos Estados Unidos da América do Norte, no final da década de 50, sob a denominação de PPBS (Planning Programming Budgeting System). Principais características : integração planejamento orçamento, quantificação de objetivos e fixação de metas, relações insumo-produto, alternativas programáticas, acompanhamento físico-financeiro; avaliação de resultados e gerência de objetivos;
- órgão – ministério, secretaria ou entidade desse mesmo grau, aos quais estão vinculadas as respectivas unidades orçamentárias;
- órgão central – incumbido de normatizar e coordenar a ação dos outros órgão que compõe um sistema;
- unidade gestora – unidade orçamentária ou administrativa investida de poder de gerir recursos orçamentários e financeiros próprios ou sob descentralização;

- unidade orçamentária – o segmento da administração direta que o orçamento da união consigna dotações específicas para a realização de seus programas de trabalho e sobre os quais exerce o poder de disposição;
- execução orçamentária da despesa – utilização dos créditos consignados no orçamento geral da união e nos créditos adicionais, visando à realização dos subprojetos e/ou subatividades atribuídos às unidades orçamentárias;
- dotação – limite de crédito consignado na lei de orçamento ou crédito adicional, para atender determinada despesa;
- empenho da despesa – ato emanado de autoridade competente, que cria para o estado obrigação de pagamento pendente ou não de implemento de condição, garantia de que existe o crédito necessário para a liquidação de um compromisso assumido; é o primeiro estágio da despesa pública;
- liquidação da despesa – verificação do direito adquirido pelo credor, tendo por base os títulos e documentos comprobatórios do respectivo crédito;
- pagamento – último estágio da despesa pública. caracteriza-se pela emissão do cheque ou ordem bancária em favor do credor;
- crédito orçamentário – autorização dada pela lei orçamentária para aplicação de determinado montante de recursos, discriminado conforme as classificações [51].

Alguns dos termos anteriormente pontuados referem-se especificamente ao orçamento da união, entretanto vale ressaltar que os mesmos aplicam-se normalmente a todas as esferas do poder executivo que são regidas pela Lei 4.320. Em nosso estudo, adotamos como requisito a observação do emprego de elementos considerados gerais entre as esferas de poder para que o mesmo possa ser validado em qualquer uma delas.

Esses termos representam apenas o universo no qual nosso estudo de caso terá atuação. O emprego dos mesmos se dará de forma simplificada e, em alguns, casos de forma abreviada ao termo apresentado na terminologia oferecida anteriormente.

Pretendemos assim, evitar o desvio do foco de nossa abordagem para detalhes técnicos que envolvam conceitos mais amplos de outras áreas da computação como engenharia de software, citada em [49]. Nosso objetivo é simplificar e tratar cada elemento constituinte do estudo de caso como unidade concreta, semanticamente definida e representativa de uma propriedade concreta do

orçamento. De modo simplificado, cada entidade do exemplo deve ser encarada como atributo atômico do orçamento, constituído de nome, significado e valor.

5.3.2 – Modelo de divulgação do orçamento

O modelo a ser adotado para a divulgação do orçamento público aplica-se a qualquer das quatro esferas do poder executivo nacional : a União, aos Estados, aos Municípios e ao Distrito Federal, pois tem como base elementos simples que estão presentes, conforme designado pela Lei No. 4.320/64, como suas normas gerais.

Nosso objetivo ao elaborar este modelo é permitir ao usuário em potencial do mesmo, obter a seguinte resposta : quanto o governo planejou de orçamento para a área X no começo do exercício e quanto concretamente foi gasto nesta área até a presente data ?. Em português popular, esta pergunta poderia ser formulada do seguinte modo : quanto foi planejado pra gastar em saúde e quanto foi gasto até agora ?. Tomando-se a saúde como exemplo.

A formalização do modelo de divulgação a ser implementado por nosso estudo de caso deve produzir o seguinte resultado, ilustrado pela Figura 5.3.2-1 :

ÓRGÃO	CRÉDITO ORÇAMENTÁRIO(A)	PAGAMENTO (B)	DIFERENÇA (A-B)
SAÚDE	1.000.000	850.000	250.000
EDUCAÇÃO	800.000	750.000	50.000
SEGURANÇA	900.000	600.000	300.000
ASS.SOCIAL	505.000	305.000	200.000
MEIO-AMBIENTE	230.000	130.000	100.000
ADMINISTRACÃO	300.000	280.000	20.000

VALORES EM REAIS, CENTAVOS OMITIDOS,
POSIÇÃO EM : 19/12/2003, AS: 15:00
FONTE : www.prefeitura.uf.gov.br

Fig. 5.3.2-1 tela de consulta para a divulgação do orçamento público via Web Services em Java

Este deve ser o aspecto do resultado final da aplicação de Web Services na divulgação do orçamento público que deverá interagir com o mesmo e disponibilizar para a rede interna do Estado (sentido de Ciência Política, conforme citado em [52]) as informações anteriormente exemplificadas.

Deve-se observar no modelo proposto para a divulgação, existe interação com os dois pontos de extremidade do orçamento público : o que foi planejado para ser investido em cada área e o que foi, concretamente investido, de forma a ser transformado em pagamento. Nosso intenção foi

demonstrar, de modo simplificado, como os dados podem ser reunidos e transformados em informação de fácil compreensão. A simplicidade do modelo possibilita a aplicação da tecnologia de Web Services e sua real e concreta utilidade.

5.3.3 – Esquema do modelo proposto

De posse do modelo proposto, passamos a descrever esquematicamente seu funcionamento no contexto, ainda genérico, da execução do orçamento público.

Inicialmente vamos considerar apenas um órgão e sua execução orçamentária atuando de forma interna e tendo como extensão nosso componente de divulgação do orçamento público. A Figura 5.3.3-1 esquema de divulgação do orçamento público via Web Services em Java, simplificado, demonstra como o esquema deverá ser entendido. Primeiro temos o órgão A executando seu orçamento normalmente com suas três operações básicas ordenadas : empenho; liquidação e pagamento. Nosso componente de Web Services comunica-se, empregando classes Java com o orçamento e verifica a quantia orçada para o exercício. Esta informação preenchera a coluna A da Figura 5.3.2-1.

O processo de pagamento, que concretiza a execução do orçamento, constitui-se a última parte. Nesta fase, nosso componente Web Services extrai informações relativas aos pagamentos realizados pelo órgão A, para preencher a coluna B, da Figura 5.3.2-1.

Ao nos referirmos ao componente de Web Services, estamos na realidade abstraindo e simplificando toda a estrutura do site necessária para a implementação do Web Services em si. Pois para que haja a comunicação entre os dados armazenados no orçamento, em sistemas de banco de dados e a posterior disponibilização destes dados para compor o Web Services, faz-se necessário a disponibilização de um site no qual será montado a estrutura de resposta as solicitações feitas ao Web Services dentro dos padrões da Internet HTML, SOAP e XML. Para nosso estudo de caso, estes padrões serão disponibilizados via tecnologia Java e suas interfaces adequadas ao modelo de Web Services aqui definido.

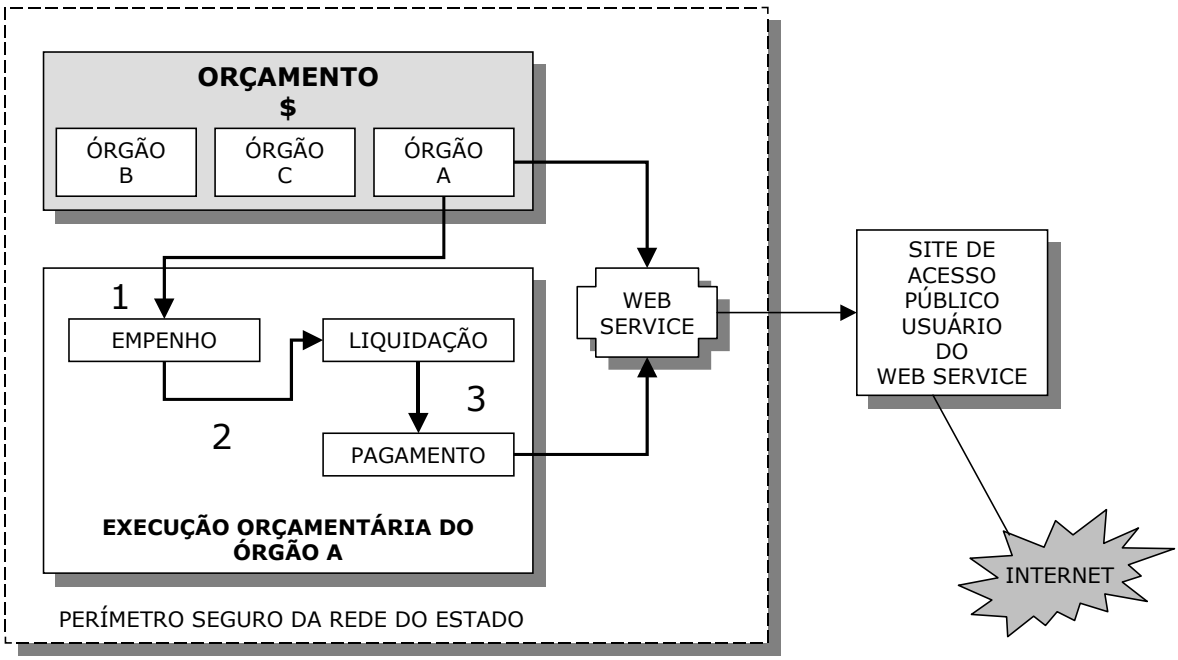


Fig. 5.3.3-1 esquema de divulgação do orçamento público via Web Services em Java, simplificado.

5.4 – Cenário do estudo de caso

Para o emprego de Web Services na divulgação do orçamento público, elegemos de modo natural a esfera de poder público que temos maior conhecimento e vivência. Tomamos como base à Administração pública Municipal. Neste cenário consideramos os aspectos relevantes que tivemos como base ao tomar para campo de pesquisa a Prefeitura Municipal de Manaus e suas características computacionais, principalmente no que diz respeito a capacidade instalada, parque de hardware e sua diferenciada rede metropolitana que interliga grande parte dos seus órgãos como pode ser observado em [58].

Em nosso estudo de caso consideramos parte d o complexo computacional disponível, elegendo pontos coerentes e selecionados especificamente para a adequação ao experimento que temos em foco, qual seja : o emprego da tecnologia de Web Services, em Java, para a divulgação do orçamento público, demonstrando que esta tecnologia apresenta-se como uma das alternativas viáveis para a integração da grande quantidade de dados produzidos na administração pública. Com a tecnologia de Web Services estes dados podem ser integrados de forma dinâmica e transformados em informações úteis no controle, no planejamento e na tomada de decisão.

Dos aspectos estudados, procuramos oferecer informações gerais de modo a permitir a formação de um cenário baseado na realidade encontrada na Prefeitura Municipal de Manaus, porém

com enfoque sempre generalista e não particularizado para esta esfera de poder, a fim de que o relatado também deva ser considerado como realidade concreta nas outras instâncias do Poder Executivo como : a União; os Estados e o Distrito Federal.

Assim, temos a ressaltar que os tópicos a seguir detalhados, ocorrem em todas as demais esferas do Poder Executivo e podem seguramente ser transladados para estas esferas, mantendo-se a devida proporcionalidade e nível de complexidade. Constituem-se pois, tópicos gerais, especialmente selecionados, visando que os objetivos de emprego da tecnologia de Web Services possa, também, ser transladada como solução factível para os demais níveis do Poder Executivo.

5.4.1 – Os órgãos do governo

Dentro da terminologia do orçamento, cada órgão dispõe de orçamento próprio que deve executar de modo a cumprir da melhor forma as ações relativas a sua área de atuação, como citado em [52]. O controle das cotas trimestrais e o planejamento setorial de cada órgão é tarefa de seu gestor. Na tentativa de desempenhar melhor estas duas tarefas, órgãos de governo adotam vários comportamentos, entre estes destacamos : adoção de sistemas manuais, sistemas localizadas de controle de partes do processo (geralmente com redigitação de partes dos dados) e solicitação para mudanças no sistema de gerenciamento central do orçamento a fim de que suas necessidades particulares sejam atendidas.

É da cultura nacional, a construção e disponibilização de sistemas de controle orçamentário, voltados para a orientação fazendocêntrica, que refletem e atendem, única e exclusivamente as necessidades do órgão de fazenda da esfera de poder na qual atuam, enquanto os demais órgãos, suas peculiaridades, necessidades setoriais, visão de orçamento sob a perspectiva de atuação de suas áreas, são simplesmente ignorados. Este comportamento se reproduz de modo perene em todas as instâncias do executivo e pode ser facilmente comprovado.

Diante dessa realidade, os vários comportamentos dos órgãos na ânsia de obter "melhor" controle sobre a parte do orçamento que lhes cabe, são perfeitamente justificados e facilmente perceptíveis.

5.4.2 – Políticas de informática

De modo simples e genérico, entendemos política de informática como a instalação de regras que regulem e padronizem os procedimentos, resolvam conflitos e determinem os rumos para a informática, em linhas gerais e específicas. Isto aplica-se em qualquer nível do Poder Executivo.

Entre os muitos elementos que contribuem para o atraso da informática pública, selecionamos cinco pontos que consideramos principais e recorrentes em qualquer esfera do poder executivo :

- sucateamento dos órgãos institucionalmente responsáveis pela política de informática;
- incapacidade técnica dos órgãos institucionalmente responsáveis de impor e conduzir qualquer política que se venha a desenhar, para a obtenção de resultados políticos esperados, na velocidade que se deseja;
- interesses políticos de que a realidade caótica atual não seja alterada;
- pressa dos responsáveis por cotas do orçamento em demonstrar suas habilidades e capacidades administrativas, principalmente, no uso dos modernos recursos de informática[5];
- não há problema de falta de recursos para a implantação da política de informática, o que falta, realmente é uma política de informática, a vontade política e o "dono" da política de informática que faça com que a mesma seja cumprida, de preferência em fases e com marcos bem definidos.

Cada um dos pontos acima, representa em sua natureza problemas de alta complexidade que fogem ao escopo deste trabalho. Tendo suas prováveis soluções no espectro da multidisciplinaridade do conhecimento e principalmente da experiência humana.

5.4.3 – Heterogeneidade

A formação da massa de heterogeneidade encontrada nas esferas do poder executivo tem sua origem e nascedouro na corrida perpetrada entre ocupantes de cargos públicos para demonstrar ao governante e aos eleitores, sua habilidade utilizar informática. Como se o uso da computação representasse, de algum modo, eficácia e eficiência na solução de problemas públicos.

[5] Refere-se aos gestores públicos como secretários, ministros, diretores e etc. que para obter prestígio político junto ao governante e a população, apressam-se para implementar soluções de informática em nível local, sem preocupar-se com o todo.

Todos no executivo adotam soluções das mais variadas espécies e famílias possíveis, porém respeitando um grande guia e mestre : a Internet. Assim, a grande maioria dos órgãos, por conta e risco próprio, se interligam a Internet, com plataformas, sistemas operacionais, linguagens, bancos de dados selecionados de modo isolado.

Sem a presença reguladora de uma política de informática, descrevemos o que alguns atrevem-se a chamar de "caos". Situação que, para a tecnologia de Web Services, pode ser denominada de heterogeneidade.

5.4.4 – Redes metropolitanas

Entende o Governo Federal que a modernização fazendária e administrativa passa necessariamente pela instalação do maior número possível de computadores interligados em grandes redes metropolitanas que devem reunir o maior número possível de órgãos de uma determinada esfera de poder. Assim, operou-se no Brasil, no final da década de 90 e até a presente data a instalação de redes metropolitanas de computadores que estão a serviço da integração de dados para a administração pública. Mas uma vez, a Internet e seus padrões influenciam fortemente estas redes que são fartamente instaladas para manterem o máximo de compatibilidade com estes padrões já estabelecidos.

Na grande maioria das capitais, prefeituras demonstram grande agilidade em manter e expandir suas redes metropolitanas, conforme exemplo citado em [58], fornecendo a impressão de que a integração está acontecendo nos níveis de modernidade fazendária e administrativa.

Quanto aos Estados da União, devido ao seu tamanho, relativa complexidade, quantidade maior de tempo em investimentos de rede privada, multiplicidade de frentes de investimento e outras complexidades, torna-se difícil, complicado e impreciso, a determinação concreta e exata da existência de redes estaduais de comunicação de dados. Nesta esfera, a cada semestre, surgem novos projetos com objetivos semelhantes aos projetos já apresentados nos seis meses anteriores e que nunca são concluídos com demonstração concreta dos resultados obtidos.

5.4.5 – O fator Internet

Entre todos os fatores que influenciam o cenário para o presente estudo de caso, a Internet, constitui-se no mais forte e mais importante. O poder de atração que a Internet exerce sobre a Administração pública é fabuloso, chega a ser monstruoso. Independente de ter, saber, entender o que

vem a ser a Internet, todo e qualquer gestor público, órgão, repartição pública ou pessoa que trabalha para o governo em qualquer instância, sabe que necessita indiscutivelmente de acesso a Internet.

Com relação a figura do ordenador de despesa, como citado em [52], este deseja que sua unidade orçamentária esteja presente na Internet e possua um site, um portal ou algo ainda mais moderno, para demonstrar aos seus pares e superiores sua habilidade em usufruir dos meios tecnológicos da moda, mesmo não havendo conteúdo de interesse público a divulgar.

O resultado, ou melhor, o efeito colateral deste fenômeno, foi o melhor possível pela ótica da tecnologia de Web Services e do cenário que desejamos caracterizar. A Internet serviu como fator atrativo de padrões que convergiram e se estabeleceram como HTML, SOAP, XML. O conjunto ideal para a tecnologia de Web Services que desejamos demonstrar como uma solução, em computação distribuída, para integrar dados produzidos pelos órgãos do governo.

5.4.6 – Esquema do cenário

Reunindo os principais aspectos anteriormente abordados e a realidade dos sistemas que controlam a execução orçamentária com orientação fazendocêntrica, apresentamos na Figura 5.4.6-1 cenário para estudo de caso. Observe que temos na figura alguns dos órgãos municipais que fazem parte do modelo de divulgação proposto interligados ao sistema central de gerenciamento da execução orçamentária. Neste cenário, nosso servidor que hospeda o Web Services estará conectado ao banco de dados no qual todos os órgãos realizam suas operações de execução de orçamento ou seja : empenho, liquidação e pagamento. O servidor de Web Services fará a extração de tais informações, de modo seletivo, do banco de dados central. Todo este relacionamento ocorre dentro dos limites de segurança da rede metropolitana, ficando exposto para a divulgação pública somente o site institucional que é o usuário potencial do Web Services desenvolvido.

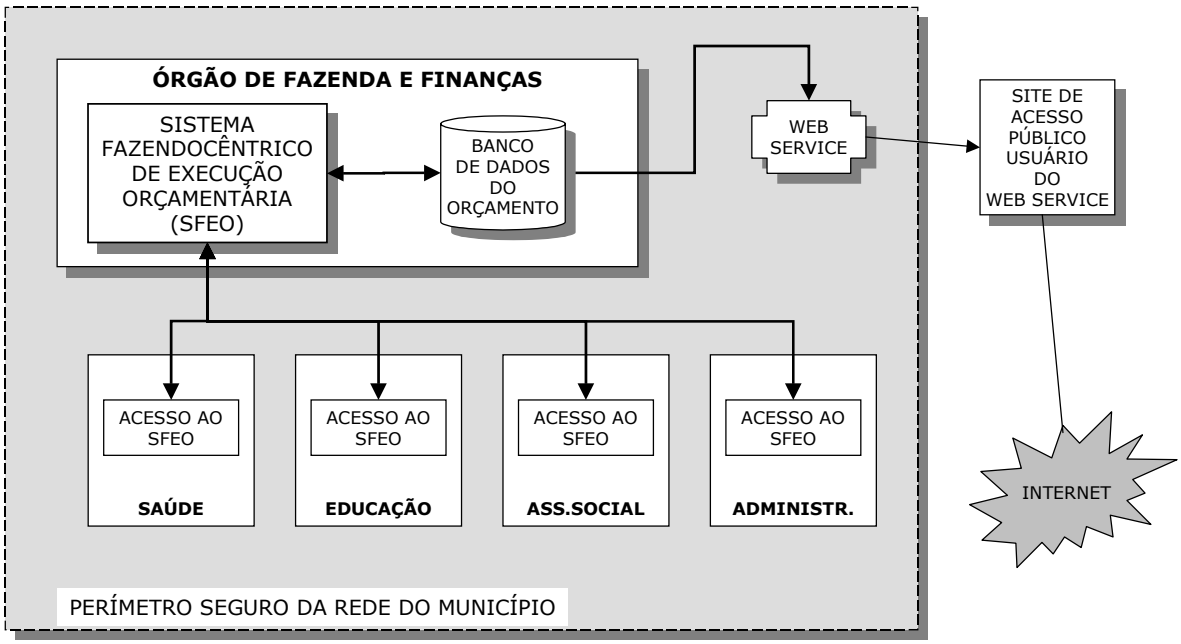


Fig. 5.4.6-1 cenário para estudo de caso.

5.5 – Aplicação de Web Services

A escolha da tecnologia de Web Services como proposta para fundamentar este exemplo baseia-se em conjunto de características que a diferenciam das demais e a tornam adequada ao ambiente em estudo, excedendo, inclusive o escopo do mesmo.

5.5.1 – Heterogeneidade

Considerando sempre o ambiente da administração pública e suas características principais, recomenda-se, como em [55], o emprego de Web Services quando se faz necessário interoperabilidade através de plataformas heterogêneas. Nesse caso quando se faz necessário a exposição de aplicações inteiras ou partes dessas para outras aplicações em plataformas diferentes com o objetivo de compartilhamento de informações, já que temos a ocorrência certa diversidade de soluções no contexto da administração pública.

5.5.2 – Baixo acoplamento

Para interação de aplicações oriundas de plataformas diferentes, em sistemas heterogêneos, que necessitam se comunicar e trocar dados, o baixo acoplamento, característica

intrínseca de Web Services, torna-se indispensável e fundamental, como citado em [1] e [3]. Temos nesta característica e na base da arquitetura orientada a serviços que rege a constituição desta tecnologia, como citado em [11], a garantia de que o emprego da tecnologia de Web Services poderá fazer com que sistemas diferentes comuniquem-se e compartilhem informações. Traduzindo-se em menor impacto em ambos os lados no caso da ocorrência de mudanças, como cita [10] que demonstra a localização dinâmica de serviços.

5.5.3 – Interoperabilidade

A interoperabilidade da tecnologia de Web Services é claramente descrita em [1] de onde citamos, de modo muito resumido, as três principais características que diferenciam Web Services dos demais modelos para computação distribuída na Internet : primeiro, emprega SOAP como protocolo de mensagens, troca mensagens em SOAP que é baseado em XML. Segundo, não tem requisitos quanto ao protocolo de transporte de mensagens, pode usar os padrões da Internet como HTTP ou até mesmo o correio brasileiro. Em terceiro, possui a capacidade de se auto descrever. O componente WSDL permite ao potencial cliente do Web Services conhecê-lo, selecioná-lo e empregá-lo, sem que haja necessidade de interação humana.

5.5.4 – Arquitetura orientada a serviço flexível

A arquitetura orientada a serviço, indica a existência de três elementos para a implementação de Web Services : o servidor do Web Services, o cliente e o registro onde os Web Services devem ser procurados pelos clientes. Porém, esta arquitetura flexível, permite que o relacionamento cliente e servidor de Web Services, dispensem o uso do registro, como citado em [55], pois podem haver casos em que o cliente sabe onde localizar o servidor e poderá solicitar a descrição dos serviços disponíveis e empregá-los segundo suas necessidades.

5.5.5 – Imunidade a firewall

Empregando padrões de uso comum na Internet, a tecnologia de Web Services permite a comunicação entre aplicações em plataformas e redes heterogêneas, sem apresentar problemas de segurança com os dispositivos de firewall, como citado em [55]. Estes dispositivos são amplamente empregados nas táticas de segurança de rede.

5.5.6 – Adequado para redes seguras

Considerando ocorrência das redes metropolitanas, encontradas na administração pública, o emprego de Web Services torna-se adequado, devido as limitações do modelo de segurança e autenticação de acesso que ainda necessita de aperfeiçoamento como citado em [3] e [16].

Nas redes metropolitanas, consideradas seguras e com perímetro de acesso limitado a parceiros confiáveis, identificáveis e monitoráveis, Web Services encontra o ambiente ideal para expandir-se e tornar-se o modelo de computação distribuído a ser adotado com menor impacto sobre as aplicações hoje existentes, como descreve [4].

Para a realização de transações seguras existem algumas propostas que de modo geral recorrem a criptografia aplicada aos conteúdos do envelope SOAP conforme citado nas Seções 3.6.6, 3.6.7 e 3.6.8.

5.5.7 – Web Services a partir de aplicações legadas

Baseada em tecnologias estabelecidas na Internet, que servem como base de padronização, a tecnologia de Web Services emprega TCP/IP, SMTP, HTTP, XML, SOAP, WSDL e outros, o que a torna capaz de, a partir da construção de camadas de software para comunicação entre sistemas legados, denominados de sistemas empresariais de informação (EIS – Entrerprise Information System) disponibilizar partes destes sistemas como Web Services para consumo na Internet, como citado em [5].

5.5.8 – Web Services aplicados a administração pública

Temos a ressaltar que todas as características elencadas anteriormente para a tecnologia de Web Services podem ser obtidas e implementadas em qualquer plataforma, linguagem, sistema operacional ou modelo de computação, desde que estes elementos sejam compatíveis e tenham aderência aos padrões estabelecidos pela Internet, como citado em [2]. Este constitui-se em dos principais fatores de expansão e aceitação desta tecnologia pela indústria.

Assim, o emprego da tecnologia de Web Services no cenário da Administração pública pode ser melhor considerado a partir da Figura 5.5.8-1 emprego de Web Services na Administração

pública Municipal, onde observamos sua aplicação a partir da disponibilização das informações setoriais produzidas por cada órgão, as quais poderão ser integradas e acessadas de modo simplificado por todos os integrantes da rede metropolitana em seu perímetro seguro e, quando conveniente, disponibilizadas para acesso público via site.

Como pode ser observado, na disponibilização de Web Services, temos uma aplicação construída com a tecnologia cliente/servidor que acessa os Web Services disponíveis para integrar os dados obtidos a partir destes, com seus próprios dados.

O acesso entre órgãos que disponibilizam Web Services na rede metropolitana é perfeitamente natural e deve ser incentivado, para maior integração de dados e produção de informações. Entretanto, esta possibilidade não foi apresentada na citada ilustração como forma de torná-la mais clara e inteligível no que diz respeito ao objetivo principal desta, qual seja, demostrar as várias possibilidades de Web Services produzidos localmente serem divulgados dentro da rede metropolitana considerando o perímetro seguro e a divulgação pública na Internet.

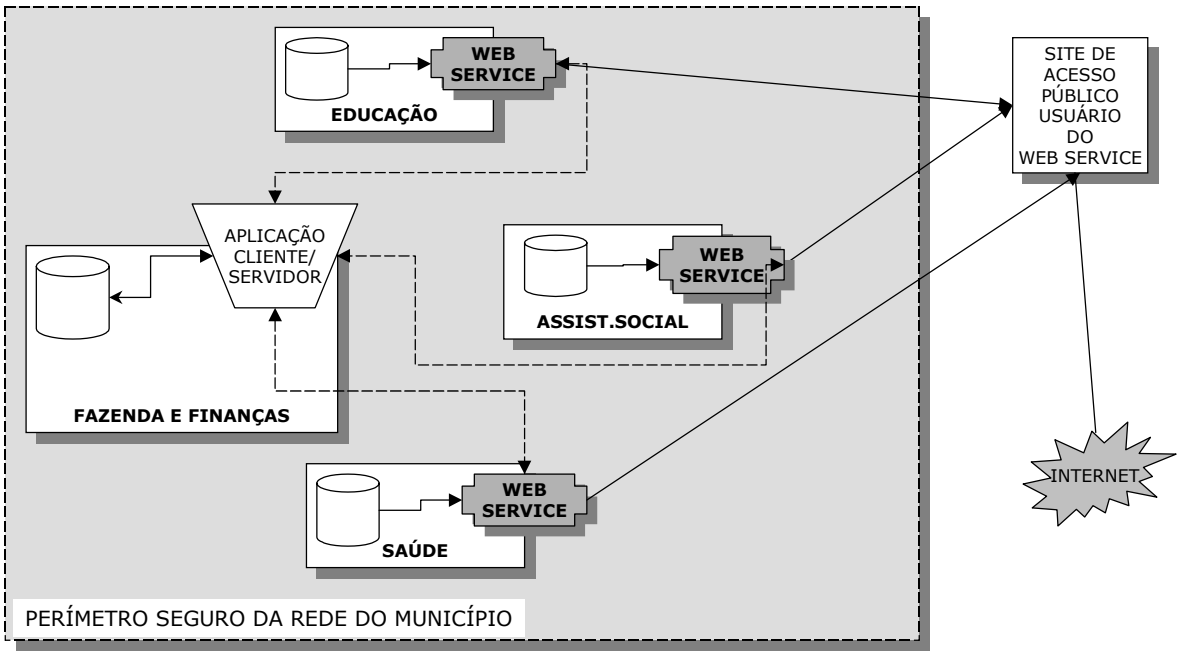


Fig. 5.5.8-1 emprego de Web Services na administração pública municipal.

Da ilustração temos a observar o aspecto fundamental da implementação que implica na instalação de estruturas segmentárias, em cada órgão, para a disponibilização de Web Services setoriais na divulgação, disponibilização e integração das informações e dados produzidos localmente. Com acesso facilitado a todo o contexto da rede metropolitana municipal, protegida pelo perímetro seguro. E como conseqüência da adoção da estrutura ilustrada, teremos a geração de três necessidades básicas :

hardware para hospedar os Web Services, software adequado para a implementação dos mesmos e pessoal com capacitação nesta nova tecnologia.

Outra alternativa a ser proposta, é a centralização de todos os Web Services locais, produzidos pelos órgãos participantes da rede metropolitana, no perímetro seguro, em um único e central servidor de Web Services, que hospedaria os serviços de todos, aproveitando-se da estrutura de interligação entre órgãos já existente. Neste esquema um órgão poderia hospedar seus Web Services no servidor central e consumir Web Services hospedados por outros órgãos.

Ao adotar a estrutura de um servidor central de Web Services para rede metropolitana, observaremos uma topologia lógica para o serviço de Web Services no formato estrela e temos a considerar como desvantagem a dependência de um único ponto para a obtenção e disponibilização de Web Services.

No que se refere a vantagens, teríamos ainda as mesmas necessidades da proposição anterior, só que em escala menor, de hardware, software e pessoal capacitado para a nova tecnologia de Web Services. A Figura 5.5.8-2 emprego de Web Services na administração pública municipal, proposta centralizada, pretende ilustrar os principais aspectos desta possibilidade.

Este arranjo tem a seu favor características como : gerenciamento centralizado de Web Services públicados; alta velocidade de implementação e disponibilização; necessidade de treinamento intensivo de apenas uma equipe para o nó central; baixo custo de instalação, possibilitando estratégias redundantes para o nó central tornar-se resistente a um grande número falhas; gerenciamento centralizado de segurança e acesso a Web Services, permitindo o acesso a estes para parceiros que estejam fora do perímetro seguro.

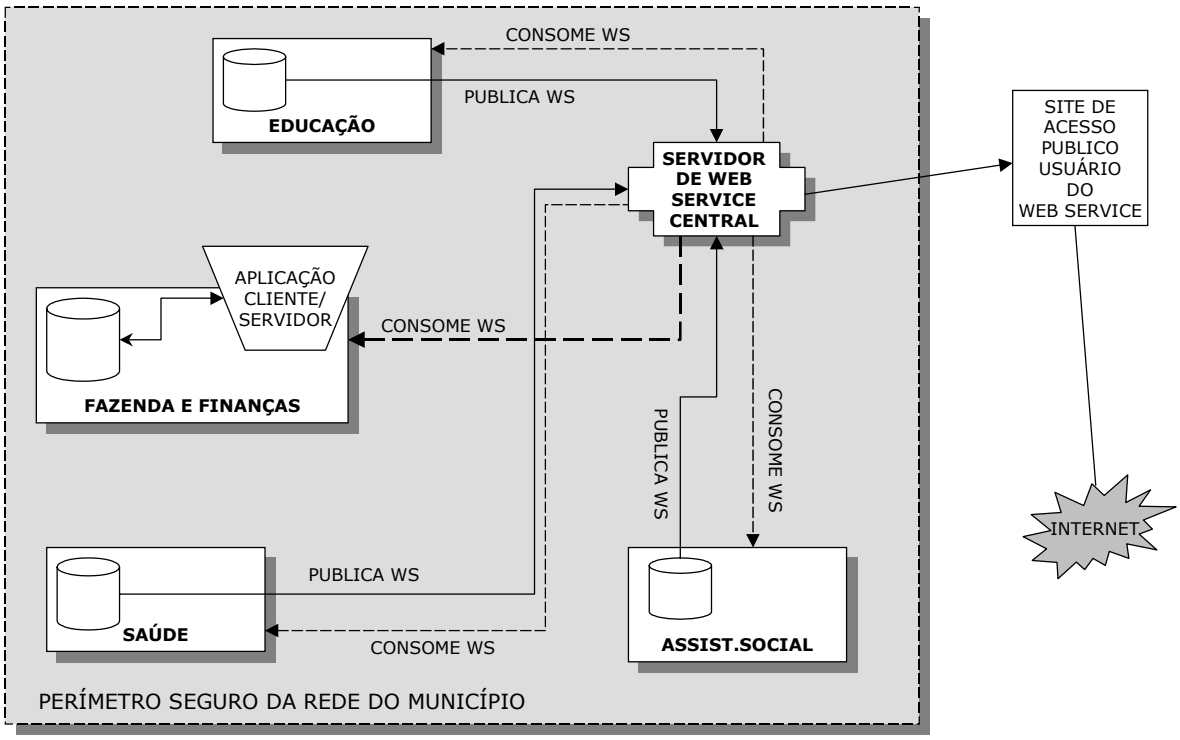


Fig. 5.5.8-2 emprego de Web Services na administração pública municipal, proposta centralizada.

Baseado na topologia estrela, o esquema proposto na Figura 5.5.8-2, necessita de cuidados especiais quanto a tolerância a falhas. Pois devido a centralização de todos os Web Services em um único nó da rede, uma falha neste nó tornaria os mesmos indisponíveis. A solução seria redundar o nó central servidor de Web Services garantindo assim pelo menos um estágio de contingência para o caso de falhas.

5.6 – Web Services em Java na divulgação do orçamento público

Para atender as necessidades da computação distribuída, obtendo vantagem competitiva da infra-estrutura e da padronização disponibilizada pela Internet, a tecnologia Java dispõe da plataforma J2EE (Java 2 Platform, Enterprise Edition) que constitui-se de mecanismos funcionando em conjunto, denominados de componentes, containers e conectores, conjuminados agrupados de tal modo, a oferecer infra-estrutura, facilidades, confiabilidade, gerenciamento e flexibilidade para suportar vários modelos de programação destinados a construção de aplicações adequadas ao ambiente de computação distribuída para Internet, ao que se tem convencionado chamar de servidor de aplicação como citado em [35] , [36] e [37].

5.6.1 – Baixo acoplamento e expansão da base de clientes

Diante da nova tecnologia de Web Services, a plataforma J2EE considera esta tecnologia como importante componente, principalmente devido sua característica natural que impõe baixo acoplamento para a comunicação entre sistemas de informações. Tal característica torna-se ideal para a integração entre sistemas de informação de empresas diferentes, dispares, baseados em plataformas diferenciadas e que, sofrem evoluções constantes, como citado em [55].

Através da absorção da tecnologia de Web Services pela plataforma J2EE, os recursos já disponíveis nesta plataforma, como aplicações e serviços, anteriormente já construídos dentro da tecnologia Java, poderão expandir-se para provê serviços, já existentes, destinados a uma ampla variedade de clientes em potencial que poderão acessar tais recursos com baixo acoplamento, através dos padrões de Internet, em qualquer plataforma [55]. Em outras palavras, a incorporação de Web Services a plataforma J2EE expande de forma inimaginável a quantidade potencial de clientes desta plataforma em si.

5.6.2 – Modelo de processamento

Servidores J2EE podem misturar várias maneiras de interação e modelo de processamento para atender a demanda de Web Services. Podem optar por comunicação síncrona e assíncrona. No que diz respeito ao modelo de processamento, podem disponibilizar Web Services orientados a documentos e orientados a chamadas de procedimentos (orientados a RPC). Entretando a grande maioria das demandas sobre Web Services recai em alguma combinação destes padrões que podem misturar-se com as possibilidades de comunicação. Porém, mesmo a mais pura implementação de Web Services que esteja baseada e orientada a chamada de procedimento, estará sempre empregando, para o servidor e para o cliente a troca de mensagens em SOAP e este será sempre uma extensão da linguagem XML [55].

5.6.3 – Arquitetura de interação

A arquitetura de interação para Web Services na plataforma J2EE foi definida de acordo com os modos de comunicação síncrono e assíncrono. Para Web Services síncronos, a arquitetura emprega a API JAX RPC tendo como ponto de acesso, ou de entrada, um servlet que funciona dentro do container. Após receber a solicitação do Web Services o servelet a encaminha para o ambiente propiciado pelo servidor J2EE [55].

No caso de Web Services assíncronos, estes também possuem como ponto de entrada ou ponto de acesso servlet que devem estar ligado, via interface JAX M a um provedor de fila, para que se processe a comunicação assíncrona, como indicado em [11].

5.6.4 – Integração de sistemas legados com Web Services

A medida que a tecnologia de Web Services adquire adesões como a da plataforma J2EE, suas aplicações aumentam. Com os servidores de aplicação J2EE, que já possuem a tecnologia testada da arquitetura de conectores para a construção de componentes em container J2EE, com capacidade de acessar e utilizar recursos de sistemas empresariais de informações legados e construídos em tecnologias anteriores a Internet [55]. Torna-se natural que a junção desta tecnologia de conectores, containers e Web Services possam transformar partes destas aplicações legadas em Web Services úteis para outras aplicações que necessitam de informações mantidas nestes sistemas legados [55].

5.6.5 – Independência e portabilidade

Mantidos todos os pactos e compromissos de padronização e independência para a linguagem Java. Temos a considerar que Web Services são independentes de plataforma. Isto significa que Web Services podem ser desenvolvidos em um grande número de linguagens para funcionar sobre muitas plataformas. A linguagem Java e a plataforma J2EE disponibilizam facilidades e vantagens para a construção, publicação e manutenção de Web Services. Os principais e mais importantes benefícios e evoluções oferecidos pela linguagem Java e pela plataforma J2EE são a independência de vendedor e a portabilidade, no que se refere a aplicação [11].

Assim, aplicações construídas sobre a plataforma J2EE podem ser publicadas e disponibilizadas sobre implementações de um grande número de vendedores. Empregando-se a linguagem Java para desenvolvimento de Web Services podemos adquirir o benefício da independência de fornecedor e ainda mantermos o adicional da independência de plataforma que constitui-se parte inerente a natureza da tecnologia de Web Services [11].

5.7 – Conclusão

Nosso objetivo foi demonstrar as principais características da execução do orçamento público no Brasil. Partindo do Plano Plurianual que fornece a diretriz para a elaboração da Lei de Orçamento, apresentada pelo poder executivo e aprovada pelo poder legislativo a cada início de ano fiscal, passando então para a dinâmica da execução do orçamento em suas três partes principais : empenho, liquidação e pagamento que ocorrem em todos os órgãos ou unidades orçamentárias citados na Lei de Orçamento.

Para o contexto dinâmico e diversificado da execução do orçamento, propomos modelo de divulgação deste que acompanha a referida cinética, a partir da extração de duas informações. As quais, quando comparadas, permitem ao cidadão leigo, verificar se as promessas de investimento anual estão ou não, sendo cumpridas. Este modelo extrai a quantidade de recursos orçados e a quantidade de recursos efetivamente pagos para todas as áreas de atuação do poder público e disponibiliza esta informação como um Web Services que poderá ser acessado por qualquer plataforma de computação aderente aos padrões da Internet. Tornando a execução do orçamento público democrática e transparente.

Concluímos demonstrando as principais características encontradas nas redes metropolitanas operadas pelo poder público e de como Web Services em Java podem ser empregados para oferecer uma solução eficiente e eficaz neste contexto de modo a implementar concretamente o modelo anteriormente proposto.

A implementação do modelo proposto, utilizando software de domínio público, nos aspectos de fornecedor e consumidor do Web Services constitui-se o tema principal da seqüência do nosso estudo de caso. São demonstrados detalhes de implementação para ambos os casos, acompanhados de esquemas e ilustrações que permitem ao leitor entender como funcionam Web Services e os mecanismos empregados para a sua publicação e consumo utilizando-se a tecnologia Java.

CAPÍTULO 6 –Protótipo para Divulgação do Orçamento Público

6.1 – Introdução

A disponibilização da tecnologia de referência e implementação para Web Services através de Java por intermédio de algumas empresas e organizações de padronização, possibilita a concretização e o oferecimento de demonstrações que podem servir como ensaios do emprego de tais tecnologias, permitindo avaliações de funcionalidade e mensuração de complexidade mais precisa em cenários como o estudo de caso aqui proposto.

Para o desenvolvimento deste protótipo selecionamos, entre várias opções disponíveis, o kit de desenvolvimento fornecido pela Sun Microsystens denominado Java WSDP (Web Services Developmente Pack 1.3), que é composto por todos os aspectos necessários a implementação de Web Services empregando-se a tecnologia Java. Neste software são encontrados os componentes fortemente abordados das APIs Java da família JAX , abordados no Capítulo 4.

Além das APIs da família JAX , este software, produto da adaptação do container Tomcat para funcionamento com Web Services, traz ainda coleção completa de tecnologias Java para desenvolvimento de aplicações Web que por estarem inclusas neste pacote, podem usufruir e acessar também Web Services fornecidos em Java e por outras tecnologias, plataformas e linguagens.

Durante a construção do protótipo, empregando o Java WSDP, constatamos que este ambiente de desenvolvimento pode ser usado para a criação de clientes Java para Web Services, bem como para o fornecimento de Web Services criados em Java. Dedicamos, porém, tópico exclusivo e discorremos unicamente sobre aspecto deste software que destina-se a tecnologia de Web Services e ao seu relacionamento com Java, ignorando as demais facilidades e avanços oferecidos pela tecnologia Java para a construção de aplicações Web.

Além do software da Sun, também empregamos outros softwares necessários ao desenvolvimento normal de aplicações, como editor de texto capaz de sinalizar as características de arquivos Java, XML, HTML [59] e outros, banco de dados compatível com os ambientes geralmente encontrados nas várias esferas de governo e ferramentas de apoio, fornecidas pelo próprio Java WSDP, como o importante ANT [25] que desempenha a função de executor de tarefas que compõe o ciclo de produção de publicação (deploy) de código em Java.

Este estudo de caso engloba as perspectivas do fornecedor e do consumidor de Web Services empregando a tecnologia Java. Seguindo este objetivo, apresentamos o detalhamento dos principais aspectos de desenvolvimento do Web Services denominado dopservice, acrônimo das palavras divulgação do orçamento público, mais a palavra Services, serviço em inglês, que fornece aos seus consumidores em potencial dois serviços possíveis. Abordamos pontos relevantes do desenvolvimento, ressaltando as características do Web Services, sua constituição, partes importantes do código fonte utilizado, estrutura e arquitetura com relacionamento entre os módulo e oferecemos algumas das principais telas obtidas quando de sua publicação (deploy).

O aplicativo Java Web destinado a consumir os serviços produzidos pelo Web Services dopservice, denomina-se webclidop, acrônimo das palavras wed cliente e dop (divulgação do orçamento público). Neste aplicativo adotamos como linha de orientação a simplicidade e a sofisticação. De aparência simples, este software emprega o acesso a Web Services e o poder da tecnologia disponibilizada por XML, utilizando aspectos avançados para o processamento de documentos XML recebidos do Web Services. Relatamos o desenvolvimento do webclidop, ressaltando a arquitetura adotada, o relacionamento entre os módulos e suas respectivas descrições, o processamento XML em combinação com aspectos avançados de JavaServer Pages, apresentamos partes do código fonte no qual o Web Services dopservice é acessado e disponibilizamos as telas principais de interface com o usuário.

Completamos nossa demonstração avaliando o cenário de aplicação da mesma e o ciclo de desenvolvimento e publicação (deploy) do código construído e disponibilizado.

No tópico Limitações reunimos pontos considerados relevantes sobre o manuseio e emprego do Java WSDP. Entre esses : disponibilidade física de tempo e condições de instalação para testes de maior complexidade e volume; a constatação árida entre o que é descrito e a realidade encontrada ao decidimos aplicar conjunto de tecnologias em um cenário real.

6.2 – Ambiente de desenvolvimento da demonstração

O desenvolvimento do protótipo funcional da aplicação de Web Services para a divulgação do orçamento público, em Java, tem como objetivo localizado o emprego e a demonstração do maior número possível das tecnologias Java disponíveis que possibilitem a percepção concreta do funcionamento destas no estudo de caso selecionado como objeto desse trabalho.

Tendo a Sun Microsystem, como a origem da tecnologia Java [11], entre as várias opções de kits e pacotes para o desenvolvimento e implementação de Web Services, em Java, adotamos o Java WSDP (Web Services Developmente Pack 1.3), fornecido em download gratuito pela própria Sun Microsystem.

O Java WSDP constitui-se conjunto integrado de ferramentas direcionadas para o desenvolvimento de Web Services composto de facilidades para a construção, montagem e publicação (deploy) de Web Services de nível e complexidade de funcionalidade básica na plataforma Java [11] , sem que haja a necessidade da instalação e integração com outros produtos ou tecnologias anteriores a essa.

Outros fornecedores oferecem kits e pacotes para o desenvolvimento de Web Services como Apache AXIS, empregado em algumas partes de [10] e base dos exemplos de [12] e o Systinet WASP, citado e demonstrado em [8]. Verificamos ainda que empresas como BEA System, oferecem produtos completos para a implementação e disponibilização de Web Services em escala de uso profissional, conforme demonstra [10], em um de seus exemplos.

Nossa escolha para o ambiente de desenvolvimento de nosso protótipo, recaiu sobre Java WSDP, por considerarmos, acompanhando a mesma linha de raciocínio expressa em [11] e [6], que o mesmo contém as últimas e as, muitas vezes, mais recentes versões de padronização para Java e APIs para XML com suas bibliotecas de suporte e quase sempre com reduzido tempo de execução.

Do ponto de vista da indústria de produção de ferramentas e ambientes para o desenvolvimento de Web Services, Java WSDP constitui-se um super e completo pacote da tecnologia Java destinado a XML [11]. O pacote XML disponível em Java WSDP, representa a orientação, o guia para a grande maioria dos demais integradores de software e fornecedores de ferramentas, pois o mesmo é composto pela API Java para XML mais completa e importante, o pacote JAX [11].

Assim, o conjunto que compõe Java WSDP, torna-se o ponto de iniciação mais adequado para todos, inclusive o desenvolvedor independente [11], disponibilizando o pacote de APIs JAX completo, o motor de container Tomcat, o servidor de registro para Web Services (Web Services registry server), interface gráfica de usuário para interação com UDDI (UDDI registry browser), ANT ferramenta para facilitar a montagem e compilação de programas Java (ANT build tool), ferramentas para publicação (deployment) e tempo de compilação [11]. Todos estes componentes completam o ciclo para implementação, disponibilização, registro do Web Services para ser encontrado no UDDI e produção de qualquer das três formas possíveis que um cliente Web Services tem a possibilidade de acessá-lo através da tecnologia Java.

6.2.1 – Componentes Java WSDP

Os componentes que constituem o Java WSDP fornecido pela Sun Microsystems, podem ser classificados em quatro categorias de uso e aplicação : APIs Java para XML que funcionam como referências para implementação, ferramentas de desenvolvimento e publicação (deployment) de Web Services utilizadas no lado servidor, o ambiente de execução dos Web Services (onde os mesmos são disponibilizados para serem acessados pelos clientes em potencial) e suporte para as especificações que constituem-se nas características que o Tomcat deve disponibilizar para que Web Services funcionem adequadamente [11]. As quatro principais categoria podem ser observadas na figura a seguir, Figura 6.2.1-1 componentes do Java WSDP.

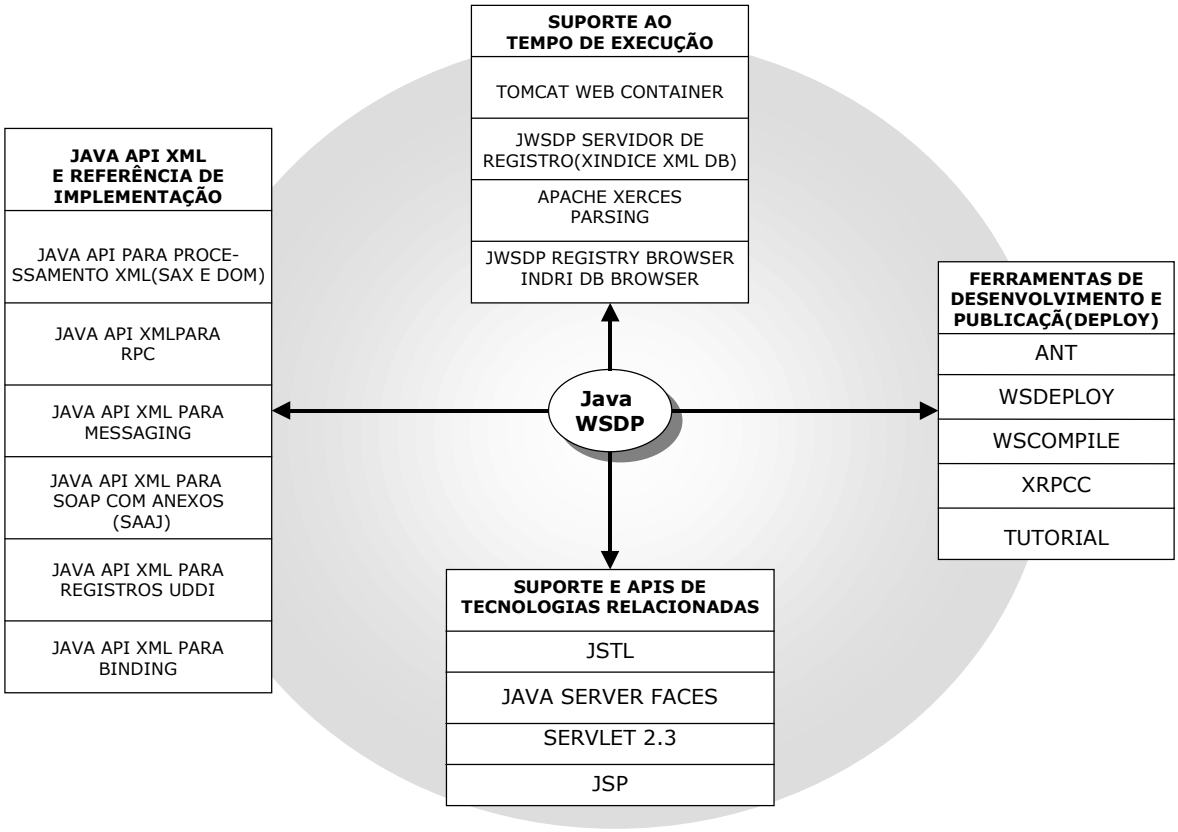


Fig. 6.2.1-1 componentes do Java WSDP (adaptado de [11]).

6.2.2 – Componentes Java API para XML

Na tecnologia Java temos a característica de componentização permeada por todos os seus aspectos e ramificações, conforme [53] e [40]. Desta forma a tecnologia de Web Services ao ser absorvida pela tecnologia Java, tem como fundamento principal o fornecimento de APIs como referência de implementação padrão de funcionamento da linguagem que devem ser compatíveis, a nível de funcionamento, com fornecedores independentes e vice versa. A seguir apresentamos as principais características das APIs que compõe o Java WSDP, conforme indicado em [11] .

- Java API for XML Processing (JAX P) - expõe o padrão XML para parsing e validação em APIs para o processamento e a transformação de XML em Java. JAX P disponibiliza suporte para DOM, SAX, XSLT e XML Schemas. O conjunto disponível é composto pelas APIs e mais o parser Apache Xerces e XSLTC. Como fica evidenciado em [11] JAX P disponibiliza método de trabalho que favorece e possibilita o emprego de qualquer implementação de parser e transformador de XML compatível com a APIs, em vez obrigar o emprego de determinadas implementações;
- Java API for XML-based Remote Procedure Call (JAX RPC) – constitui-se na API utilizada para implementar Web Services em Java, que são produtores de mensagens baseadas em SOAP 1.1 e

utilizados a partir de chamadas remotas a procedimentos. Emprega-se essa API também para o desenvolvimento de clientes consumidores de Java Web Services baseados em RPC;

- Java API for XML Messaging (JAX M) – disponibiliza abstração sobre a camada de transporte de mensagem e viabiliza aplicações Java para o envio e o recebimento de mensagens orientadas a documentos em XML. JAX M implementa e manipula SOAP 1.1 e emprega as especificações de SAAJ para fornecer suporte ao tratamento de SOAP com anexos;
- SOAP with Attachments API for Java (SAAJ) – permite a criação de aplicações orientadas a mensagens que utilizam SOAP com anexos. Inicialmente era parte integrante da API JAX M, agora desmembrada porque é utilizada com freqüência por outras APIs da família JAX ;
- Java API for XML Registries (JAX R) – tem como função oferecer proposta de padronização para a uniformização, em uma API de acesso e manutenção (inclusão, alteração, exclusão) para padrões de registros de Web Services, já estabelecidos como ebXML e UDDI;
- Java Architecture for XML Binding (JAX B) – possibilita para XML Schema (ou DTD, se for o caso), a transformação de documentos XML em classes Java equivalentes. As classes geradas manipulam os detalhes de parsing da instância XML e são aderentes as restrições do esquema [11]. Experiências independentes como em [19] e em [18] demonstram a viabilidade desta arquitetura que tende a tornar a integração e a manipulação de XML em Java mais natural e aderente.

Com este resumo, demonstramos de forma resumida, todas as APIs da tecnologia Java que estão disponíveis como referência de implementação para Web Services e foram detalhadamente estudadas no Capítulo 4 deste estudo.

6.2.3 – Ambiente de execução do Java WSDP

Como definido anteriormente pela API de referência JAX RPC e JAX M, o ponto de entrada de um Web Services em Java constitui-se necessariamente em um Servlet. Deste modo o Java WSDP traz como um de seus componentes e ambiente de execução o já bastante evoluído Tomcat Web Container, que suporta a especificação de Servlet 2.3. O funcionamento do ambiente é bastante simplificado possuindo scripts para acionar e desligar, pode ser administrado por aplicação que funciona via browser.

Um registro constitui-se parte essencial da arquitetura Web Services, baseada em arquitetura orientada a serviço, como citam [2], [3] e [11]. No ambiente de execução Java WSDP está incluso um servidor de registro de Web Services que pode ser empregado para testar programas que

utilizem a API JAX R. Este servidor de registro é compatível com o modelo de informação do padrão UDDI v. 2 e destina-se ao uso privado. Está implementado no container do Tomcat e pode ser acessado por programas escritos em qualquer linguagem, inclusive, Java. Como banco de dados de suporte, o servidor de registro emprega o banco de dados XML nativo Xindice Apache. Além do da função do servidor de registro para Web Services ainda está disponível uma interface de usuário, ium pequena ferramenta GUI (Graphical User Interface), escrita para ser usada em computador isolado (standalone) ligado a Internet [11].

6.2.4 – Ferramentas de apoio do Java WSDP

Para a construção efetiva de Web Services, o Java WSDP disponibiliza conjunto de quatro ferramentas básicas que permitem o desenvolvimento, a configuração e a publicação de Web Services. Este conjunto tem a finalidade de facilitar o desenvolvimento de Web Services em Java, mantendo detalhes da tecnologia WUST, como citado em [11], o mais distante possível da fase de construção de Web Services.

- ANT – constitui-se ferramenta extremamente versátil, a bastante tempo fazendo parte do suporte ao desenvolvimento no ambiente Java, incluindo-se aí a tecnologia J2EE. Sua principal função é auxiliar nas tarefas de compilação completa dos fontes (build) e na publicação (deploy) das aplicações [11] . O poder, flexibilidade e a versatilidade do ANT como ferramenta de apoio a compilação completa dos fontes (build) e outras tarefas configuráveis no desenvolvimento profissional em Java, torna-se evidente pelo fato de um grande número de ambientes de desenvolvimento integrado (Integrated Development Enviroments - IDE) disponibilizarem suporte para a integração opcional desta ferramenta, caso o desenvolvedor assim o queira. Mesmo o uso do ANT via linha de comando torna a publicação (deploy) dos componentes de Web Services facilitada em larga escala. A versão padrão do ANT traz tarefas já definidas e embutidas como manipulação de arquivos JAR, ZIP, CAB, copy, FTP. Existem versões direcionadas para algumas plataformas e para produtos específicos, como tarefas para Junit, EJB. No caso do Java WSDP a versão ANT que acompanha o pacote tem tarefas e scripts construídos previamente para a compilação de classes stubs e classes skeletons, além da automatizar a tarefa de publicar (deploy) o Web Services desenvolvido, no container Tomcat;
- wsdeploy – tendo como base uma aplicação a ser publicado (deploy) em um container, a tecnologia Web Services, em Java, necessita de uma ferramenta para realizar o empacotamento das partes que compõe o Web Services. O wsdeploy é empregado para criar arquivo publicável (deployable) com a extensão .WAR que conterá informação necessária para a criação e o funcionamento do ponto de

acesso JAX RPC relativo ao Web Services [11]. O arquivo .WAR deve ser publicado no Tomcat. O wsdeploy faz parte do conjunto de ferramentas que compõe o Java WSDP, porém tem a característica de funcionar de modo independente dos demais e pode ser usado independente do ANT ter sido empregado antes, sua função é realmente tomar os arquivos gerados e empacotá-los de modo que o container do Tomcat possa processá-los como uma aplicação especial de Web Services [11]. O arquivo .WAR é composto pelas classes do Web Services que residirão no lado servidor e arquivos manifest de configuração que descrevem como estas classes se relacionam e devem ser ativadas. O primeiro arquivo manifest é o Web.xml que contém os padrões de aplicação Web para container Tomcat. O segundo, jaxRPC-ri.xml, constitui-se em um caso particular específico para programas desenvolvidos sobre a API JAX RPC e são criados utilizando-se mecanismos disponíveis no Java WSDP [11].

- wscompile – para a produção de stubs RPC, que funcionam do lado cliente empregamos a ferramenta wscompile que toma toda a implementação em Java e produz as classes necessárias ao lado servidor do Web Services, o WSDL e opcionalmente os stubs para o lado cliente. Essa ferramenta também foi criada para que, de posse de arquivo WSDL, o desenvolvedor Java produza as classes stubs RPC necessárias para acessar o Web Services descrito no WSDL, sem ter que se importar sobre qual linguagem o Web Services foi desenvolvido, ou com outro detalhes mais internos [11]. Essa ferramenta produz as classes Java necessárias para acessar Web Services como se fossem classes Java nativas[11].

- Registry Browser – o Registry Browse constitui-se em programa com interface gráfica, GUI, escrito em Java, que pode ser empregado para pesquisar, localizar e adicionar e alterar informações ao registro UDDI que acompanha o Java WSDP, bem como qualquer outro registro UDDI que permita conexões via Internet. O programa implementa a aplicação da API JAX R como cliente e pode-se perceber como um registro funciona a partir da visão do cliente que o consulta via programa isolado (standalone) em um computador, através da Internet [11].

Na Figura 6.2.4-1 ferramentas para o desenvolvimento de Web Services empregando Java WSDP, ilustramos, considerando os detalhes principais, o modo mais convencional, dentro da lógica indicada pelo Java WSDP, para se utilizar as ferramentas. Observa-se que o processo ilustrado é composto de seqüência ordenada e que cada ferramenta comporta-se como um processo, coordenado, de certo modo, pela ferramenta ANT.

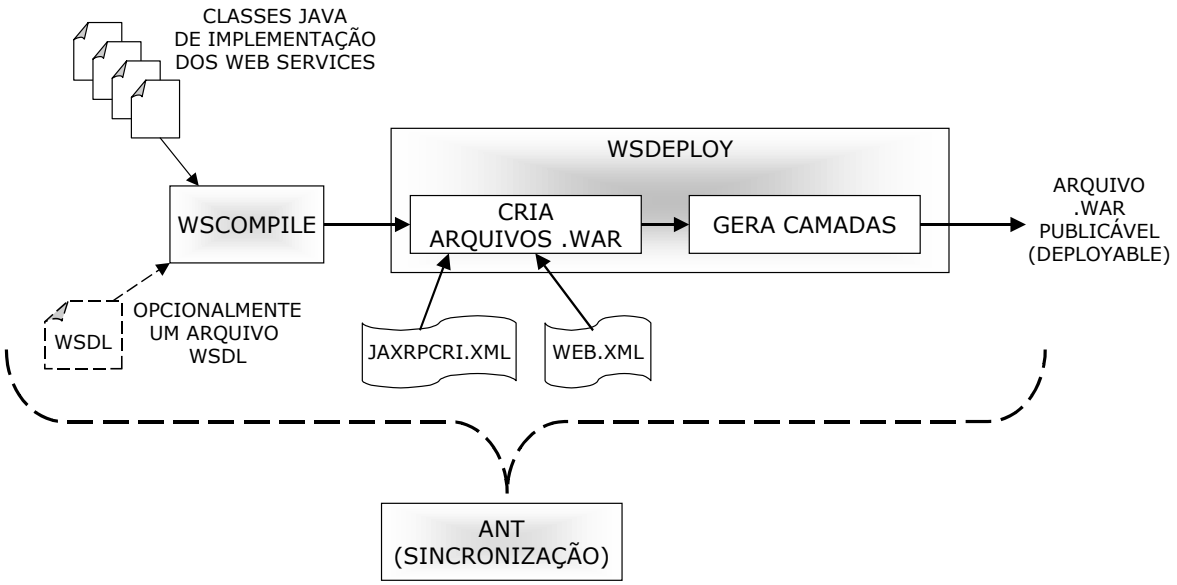


Fig. 6.2.4-1 ferramentas para o desenvolvimento de Web Services empregando Java WSDP.

6.2.5 – Especificações de suporte para Java WSDP

Mesmo não diretamente relacionados a linguagem XML e APIs que a manipulam efetivamente, o pacote Java WSDP dispõe de suporte especial para especificações relativas ao núcleo de infra-estrutura necessária as diversas modalidades de aplicações que empregam a tecnologia Web Services através de Java [11].

- Java Server Pages Standard Tag Library (JSTL) – constitui-se em conjunto comumente usado para personalizar JSP (JavaServer Pages, páginas Java que o servidor transforma em HTML) tags, um método, um modo bem formado e definido para a criação de novos marcadores (tags) e expressões na linguagem, que contribuem para simplificar o desenvolvimento de páginas;
- Java Server Faces – tecnologia que incluirá padrões do conjunto empregado em marcadores (tags) JSP, disponibilizando um método, um modo bem formado, bem definido e simplificado para a criação de formulários complexos em HTML e elementos, componentes, para HTML no ambiente de desenvolvimento JSP;
- Java Server Pages e Servlets – o Java WSDP suporta a versão JSP 1.2 e Servlet 2.3 ambos compõe o fundamento, o núcleo de tecnologias para Web Services em Java [11].

6.2.6 – Ferramentas de apoio e suporte ao desenvolvimento

Para auxiliar o desenvolvimento de construção desta demonstração empregamos ainda as seguintes ferramentas :

- Java Development Kit, Ver. 1.4.2_02;
- sistema operacional Windows XP Professional, Services Pack 1;
- editor de programas Scite que pode ser encontrado gratuitamente no site www.scintilla.org;
- microcomputador pessoal Pentium 4, de 2.0 MHZ, 512 MB RAM;
- banco de dados Oracle 9i Rel 2, cópia de demonstração em cd-rom cedido pelo fabricante;

6.2.7 – Aspectos cobertos pela demonstração

Conforme mencionando anteriormente, essa demonstração engloba o emprego da tecnologia Java em Web Services nos aspectos do servidor, fornecedor de Web Services e no aspecto do consumidor, do cliente de Web Services. No caso do fornecedor de Web Services, a implementação seguiu linhas gerais delimitadas no Capítulo 5 e corresponde a consultas realizadas ao banco de dados Oracle [60] , esquema que contém a execução do orçamento, disponibilizadas como Web Services para acesso público, obedecendo esquema de segurança estabelecido pelas redes metropolitanas, conforme comenta [58] . Nossa implementação foi direcionada a concretizar, em software, o que nas ilustrações apresentadas nas Seções 5.4.6 e 5.5.8, denominamos de Web Services e Servidor de Web Services Central. Este foi implementado, empregando-se a API JAX RPC, e disponibilizado para ser consultado por qualquer potencial cliente que siga o modelo descrito em seu WSDL, conforme preconiza [63].

Na perspectiva do cliente, consumidor de Web Services que emprega para tal a tecnologia Java, o desenvolvimento desta demonstração oferece aplicação simples onde, através da interação com o browser, o usuário de Internet, poderá obter as informações da execução orçamentária em uma forma simples e inteligível, como a definida na Seção 5.3.2. Além desta informação, também implementamos opção para que o usuário possa inspecionar o conteúdo real e concreto recebido a partir do Web Services fornecedor, com o intuito de demonstrar um pouco da flexibilidade e do poder de XML, conforme indica extensamente [12].

Assim, para maior facilidade de compreensão e entendimento do contexto sobre o qual realizamos a presente demonstração apresentamos a seguir a Figura 6.2.6-1 esquema simplificado da

demonstração de implementação do dopservice, na qual podemos perceber com clareza a presença dos componentes mencionados : fornecedor e consumidor de Web Services e a possibilidade de escalabilidade na divulgação de informações do orçamento, através da implementação de outros consumidores, que podem estar em qualquer plataforma e/ou linguagem, conforme indicam os princípios gerais da tecnologia de Web Services em [1], estes consumidores podem ser observados em linhas pontilhadas na figura e devem submeter-se à política de segurança [64] implementada pela rede que hospeda e disponibiliza o servidor de Web Services na Internet.

Em nosso exemplo ilustrado, citamos a rede metropolitana de uma cidade, pois a mesma compõe o cenário já estudado no Capítulo 5 anteriormente. Vale ressaltar que no site onde está indicada a hospedagem da aplicação consumidora de Web Services temos instalado aí o container para Servlet Tomcat, responsável por fazer funcionar a aplicação consumidora de Web Services desenvolvida para essa demonstração denominada de Webclidop, acrônimo para Web cliente do dopservice e a camada de acesso ao Web Services em Java fornecida pelo JWSDP. O usuário e o seu browser, acessam essa aplicação apontando para o servidor que a hospeda, utilizando apenas os padrões já estabelecidos na Internet.

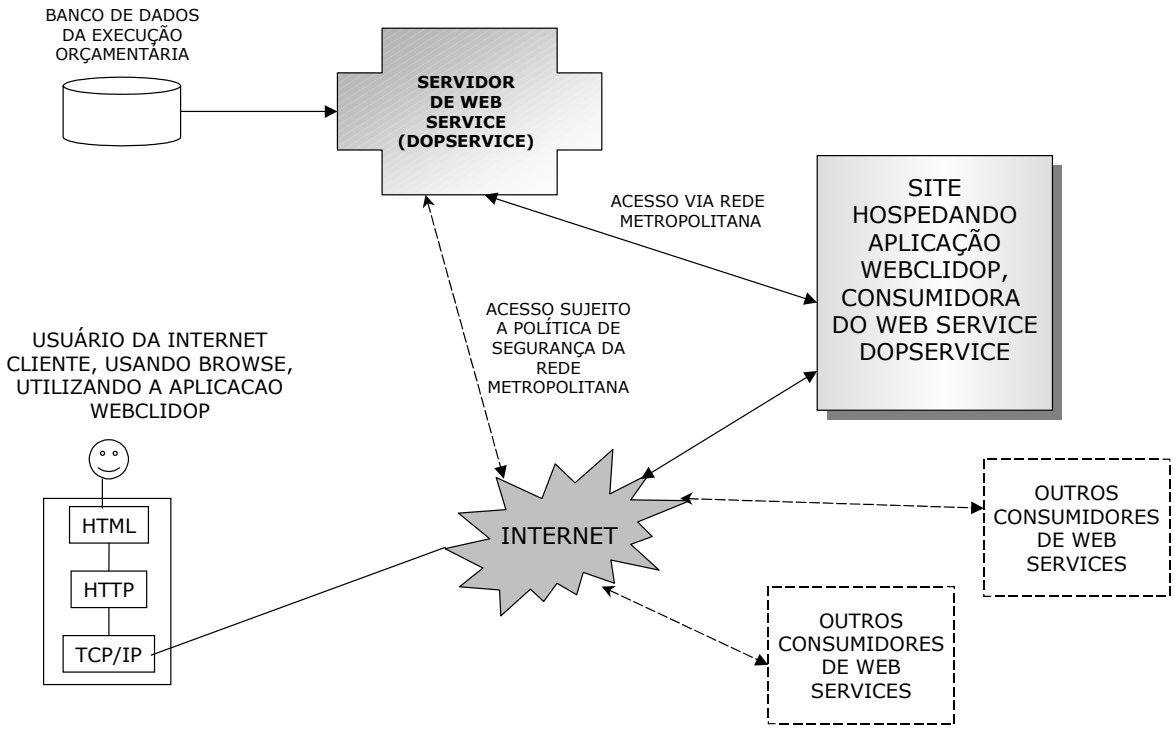


Fig. 6.2.6-1 esquema simplificado da demonstração de implementação do dopservice.

6.3 – Desenvolvimento do Web Services dopservice

Para o desenvolvimento de Web Services aplicando-se a tecnologia Java, disponível em JWSDP 1.3, pode-se percorrer vários caminhos, nossa demonstração caracterizou-se pelo uso intensivo e exclusivo da API JAX RPC.

Inicialmente projetamos um Web Services que tem como finalidade a divulgação do orçamento público, de acordo com proposto na Seção 5.3.2.

Com o intuito de demonstrar o poder e a flexibilidade da tecnologia de Web Services e do XML, aproveitamos indicação de [10] e criamos o nosso Web Services fornecendo documentos XML, de modo simples em Strings, tipo perfeitamente compatível com Java e XML. Assim, implementamos o Web Services denominado DivulgaOrcamento que pode ser acessado através da porta DivulgaOrcaIFPort e possui as seguintes mensagens, que no mundo Java serão transformadas em métodos : docxml e msgxml, onde a primeira não recebe parâmetro, sendo somente solicitada a retornar um String e a segunda recebe um número inteiro retornando apenas um String.

Para o serviço docxml, que constitui-se o alvo principal da implementação, criamos, dentro da String, um documento xml bastante interessante, com a seguinte estrutura :

```
<?xml version=1.0?>
<documento>
     <linha>
         <abrev>
            (abreviatura do órgão)
         </abrev>
         <orgao>
             (nome por extenso do órgão)
         </orgao>
         <orcado>
             (valor orçado)
         </orcado>
         <pago>
             (valor efetivo que foi pago pelo órgão)
         </pago>
         <diferenca>
             (diferença entre valor orçado e valor pago)
         </diferenca>
         <percentodif>
             (percentual da diferenca, ou seja quanto em percentual
              já foi efetivamente investido pelo órgão)
         </percentodif>
     </linha>
</documento>
```

O elemento root deste documento xml é <documento>, para cada órgão do orçamento público, há um elemento <linha> e todos os seus sub elementos. Quando a aplicação solicita do Web Services, o serviço DivulgaOrca.docxml (), recebe em resposta essa String, resultado de uma consulta ao banco de dados da execução orçamentária, a qual deve ser processada como um documento XML. E isto nós executamos com o auxílio da tecnologia Xpath no módulo que transforma este dado em informação inteligível.

A implementação do segundo serviço, neste Web Services, denominado de DivulgaOrca.msgxml (int), também segue a mesma linha, porém com muito mais simplicidade. Trata-se apenas de um serviço de teste, que de acordo com o número enviado, retorna uma String, também um documento xml, com a seguinte estrutura :

```
<?xml version=1.0?>
  <mensagem>
       (mensagem de acordo com o numero enviado)
  </mensagem>
```

Este documento é de extrema simplicidade e dispensa maiores comentários, apesar de descrito no WSDL, o mesmo não é utilizado em nenhuma das partes desta demonstração. Apenas foi citado como exemplo de extrema simplificação empregada para passos iniciais do desenvolvimento. Entretanto, sua colaboração é válida, pois nos fez constatar, na prática, que nos casos em que o serviço docxml () não funcionou ou não respondeu por problemas em sua fonte de dados, o banco de dados da execução orçamentária, este serviço, permaneceu estável e em pleno funcionamento. Fato que demonstra a robustez e a confiabilidade do container e da implementação oferecida pela tecnologia Java WSDP 1.3 testada.

A grande maioria das informações aqui fornecidas essa presente no arquivo WSDL que o Web Services dopservice disponibiliza como parte de sua implementação. Entretanto o fato de a resposta a ser enviada, "dentro" do String de resposta, deverá ser observado pelo desenvolvedor que desejar empregar este Web Services em uma de suas aplicações, fato que cabe perfeitamente ao desenvolvimento de sistemas na esfera da Internet. O envio de um documento XML bem formado dentro de uma String, constitui-se vantagem competitiva para o Web Services que poderá, facilmente agregar novas e mais interessantes informações, sem que entretanto, tenha de alterar suas definições de interface, ficando a critério dos usuários do mesmo utilizá-las ou não, como é o caso das colunas, propositadamente, não empregadas em nossa demonstração. A retirada ou inclusão das mesmas, não afeta em nada ao desempenho da aplicação cliente.

6.3.1 – Requisitos do Web Services dopservice

Os requisitos a seguir descritos constituem-se restrições técnicas e particularidades encontradas para que a implementação do Web Services projetado pudesse ser concretizada. Entre estes destacamos os de maior importância, em nosso trabalho de desenvolvimento e implementação :

- em sua instalação original o Oracle 9i, endereça a porta 8080, a mesma utilizada pelo Tomcat e JWSDP 1.3, optamos por desativar a resposta para essa porta no Oracle [61] ;
- para que o Web Services dopservice venha acessar o banco de dados Oracle, faz-se necessário a disponibilização da classe classes12.zip no diretório de bibliotecas comuns do JWSDP 1.3;
- a utilização das facilidades de container, como por exemplo o gerenciamento de pool de conexões com o banco de dados requer a configuração do Data Sources, ligando o container do Tomcat ao banco de dados Oracle; conforme indica a Figura 6.3.1-1 configuração do Data Source para Oracle no Tomcat, oferecida a seguir;
- para que as classes de implementação do Web Services tenham acesso aos Data Source configurados e disponibilizados no container do Tomcat, via tecnologia JNDI (Java Naming and Directory Information), faz-se necessário, após a publicação do Web Services, a configuração do Resource Link; a Figura 6.3.1-2 configuração do Resource Link para dopservice utilizar Data Source do Oracle via JNDI, demonstra este requisito;
- a estrutura de dados que compõe o orçamento público é bastante complexa e muitas tuplas necessitam ser processadas para que se obtenha resultados consistentes e simplificados, como os apresentados pelo Web Services DivulgaOrca.docxml (). Assim o banco de dados a ser adotado deve possibilitar o suporte a subconsultas, como indicado em [47] e [46], para que se possa obter os resultados esperados da camada de dados. Além das subconsultas empregamos ainda o agregador GROUP BY para compor o acesso ao banco de dados na classe DopDAO2, indicada na consulta em SQL (Structure Query Language) a seguir, demonstrada como instrução Java :

```
 String query =
"select orgao_abrev, f_orgao, "+
"to_char ( orcado,'999,999,999.99') as f_orcado, "+
"decode ( investido,0,' ', "+
" to_char (investido,'999,999,999.99') ) as f_investido, "+
" to_char ( (orcado-investido),'99,999,999.99') as "+
"falta_investir, to_char ( "+
"trunc (  ( ( (orcado-investido)/orcado)*100),0),'999')||' %' "+
"as falta_percentual "+
"from  ( "+
       "select orgao_abrev, rpad (orgao_extenso,80) as f_orgao,"+
       "sum (valor_orcado) as orcado, "+
```

```
"sum (valor_pago)   as investido "+
"from orc_orcamento "+
"group by orgao_abrev, rpad (orgao_extenso,80) "+
"order by 3 desc )";
```

- após, o atendimento destes requisitos a utilização da tecnologia JNDI para a implementação de Web Services, torna-se bastante simplificada, como demonstra o fragmento de código seguir, extraído da classe que realiza a abstração de dados para Web Services, denominada dopdao.dopDAO2. Observa-se o referência direta a configuração do Resource Link :

```
InitialContext ic = new InitialContext ();
Context envCtx =  (Context) ic.lookup ("Java:comp/env");
DataSource ds  =  (DataSource) envCtx.lookup
                  ("jdbc/oradop");

conn    = ds.getConnection ();
stm_bco = conn.createStatement ();
```

- a configuração de Resource Links deve ser única entre todas as aplicações do container, isto quer significar que cada aplicação deve referenciar-se de modo exclusivo, em relação as demais e em relação aos demais contextos;

- a publicação de Web Services deve obedecer rigidamente as recomendações da API JAX RPC, nenhum elemento de alocação de recurso, acréscimo de qualquer espécie é permitido ao arquivo de configuração Web.xml e jax-rpc-ri.xml os quais são fornecidos junto com a API. Tais configurações devem ser realizadas após a publicação (deploy) do Web Services.

A seguir apresentamos as figuras mencionadas anteriormente como pontos de requisitos técnicos para a implementação do Web Services dopservice. Observe que estas figuras referem-se aos painéis de configuração do Tomcat, adaptado como container para o JWSDP, conforme indica [63] e os itens configurados devem ser executados somente após a publicação (deploy) do Web Services .

Fig. 6.3.1-1 configuração do Data Source para Oracle no Tomcat.

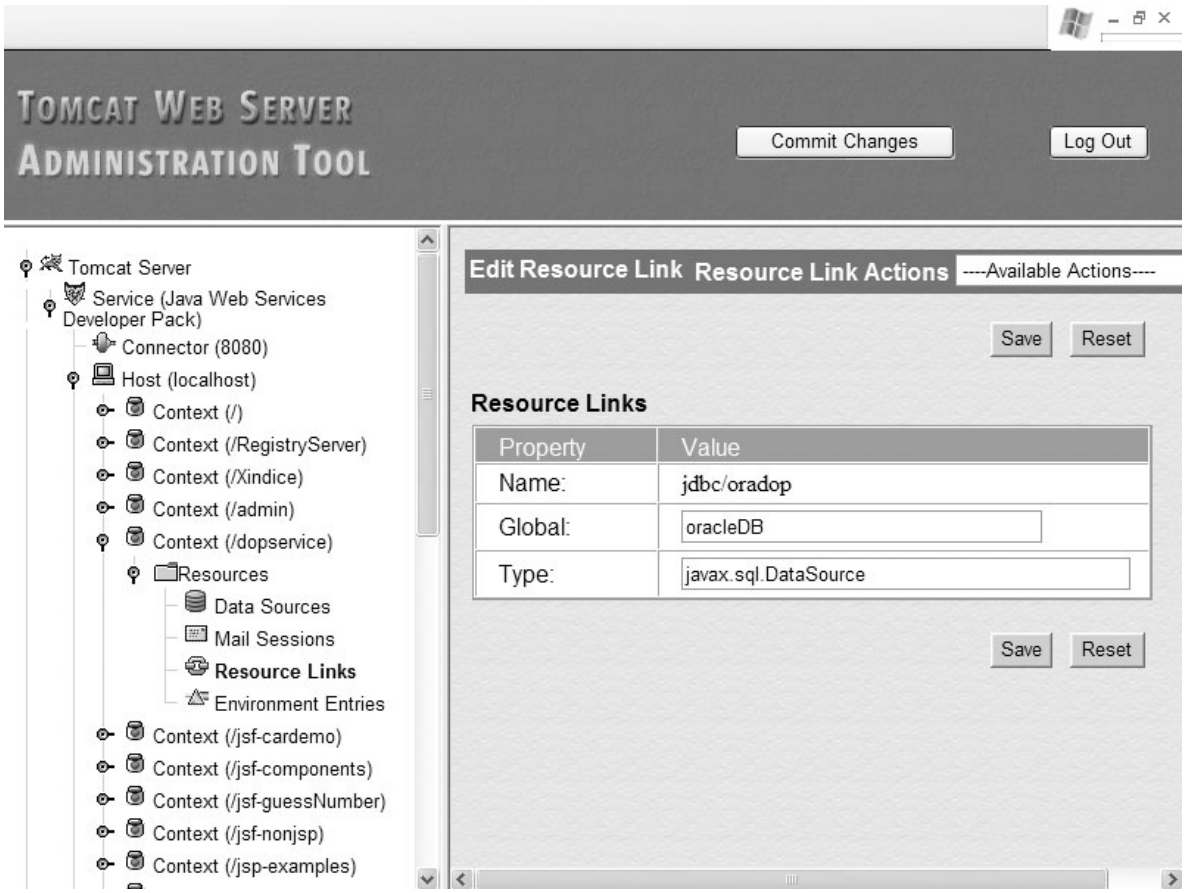


Fig. 6.3.1-2 configuração do Resource Link para dopservice utilizar Data Source do Oracle via JNDI.

6.3.2 – Fases de desenvolvimento do Web Services dopservice

Diante das várias possibilidades oferecidas pela API JAX RPC para a construção de Web Services e a forma como os utilitários wscompile e wsdeploy podem ser manuseados para prover e combinar os vários requisitos necessários a implementação de Web Services. Sintetizamos o trabalho de desenvolvimento para essa demonstração, percorrendo o caminho a seguir.

Partindo do projeto de Web Services a ser construído, já definido anteriormente, passamos a implementação das classes necessárias para a concretização do mesmo e dos demais passos necessários, como passamos a descrever :

- implementação de classes em Java, com extensão do pacote Java.rmi.remote, para interface que expõe os métodos disponibilizados como serviços dos Web Services;
- os métodos expostos na interface, devem ser efetivamente implementados por classe constituída de código para a concretização dos mesmos. Neste caso pode-se utilizar toda a flexibilidade da linguagem Java e agregar código fonte de outras classes e outros pacotes para a implementação do Web Services;
- após a compilação da implementação em Java do Web Services, podemos reunir um conjunto de arquivos de configuração e entregá-los ao utilitário wscompile. Nessa fase são produzidos dois arquivos necessários a consumação do processo, estes com nomes arbitrários, no nosso caso são : dopservice.wsdl e model.gz. O dopservice.wsdl é a descrição do futuro Web Services em WSDL especificando em detalhes como cada método exposto na interface é implementado, poderá ser acessado por seus futuros e potenciais clientes. O arquivo model.gz constitui-se a descrição do serviço para o container do Tomcat, como mencionado em [63].
- a partir dos arquivos compilados das classes em Java, dos arquivos produzidos pelo utilitário wscompile, em nossa demonstração, dospservice.wsdl e model.gz, e mais os arquivos de configuração padrão como Web.xml e jax-rpc-ri.xml, em uma estrutura de diretório específica, devemos produzir, segundo [63] , um arquivo compactado com o utilitário Java JAR que deve ter a extensão .war . Para facilitar o entendimento, vamos chamar este arquivo de pre-dopservice.war;
- o último passo para a produção do Web Services é o mais simples de todos, basta fornecer o arquivo anteriormente produzido como entrada, pre-dopservice.war para o utilitário wsdeploy que o mesmo produzirá o dopservice.war, o qual poderá ser publicado (deploy) no container do Tomcat, como um Web Services normal;

Oferecemos a ilustração a seguir, Figura 6.3.2-1 processo de desenvolvimento e implementação do dopservice, que demonstra de forma esquemática o processo anteriormente descrito, inclusive mencionando os nomes reais dos arquivos que são usados, indicados como referência de implementação por [63] e a documentação instalada juntamente com o Java WSDP 1.3. Observe ainda que essa figura detalha e demonstra, no aspecto prático e concreto o mesmo procedimento que é apenas sugerido na Figura anterior 6.2.4-1. Nessa situação, também a utilização do ANT torna-se de fundamental e indispensável importância devido ao alto grau de complexidade que as operações de compilação, wscompile e wsdeploy exigem.

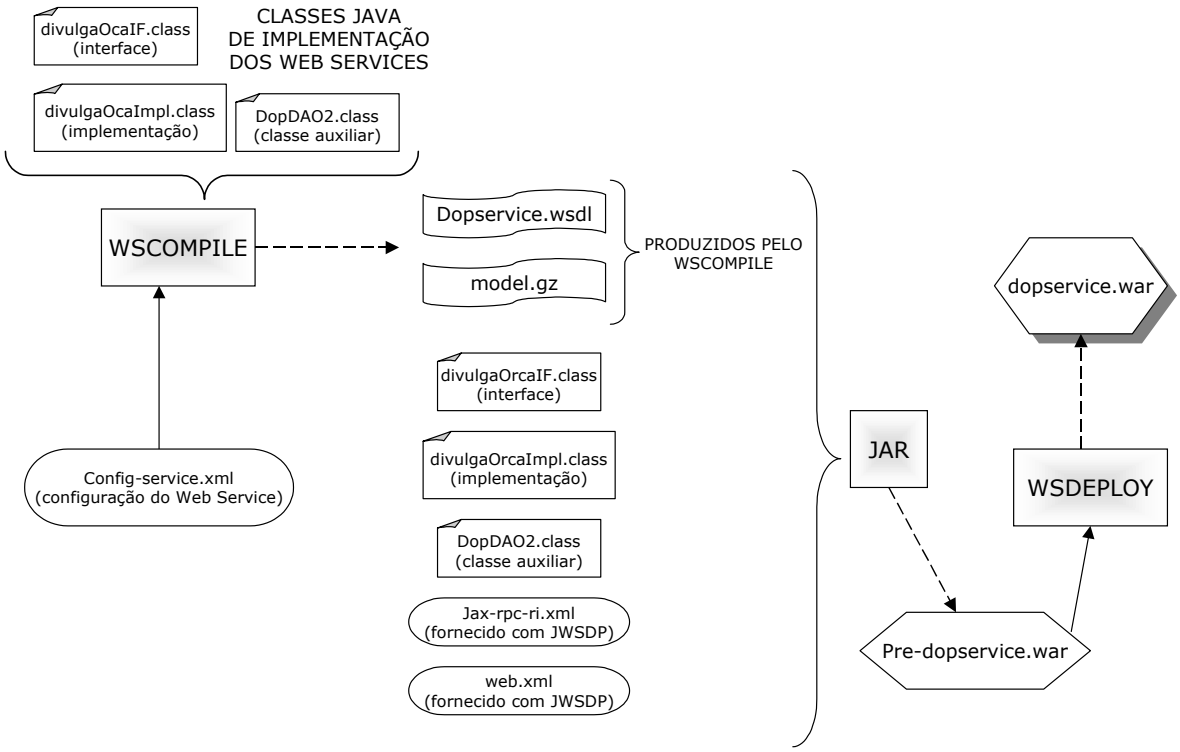


Fig. 6.3.2-1 processo de desenvolvimento e implementação do dopservice.

6.3.3 – Estrutura do software e módulos

A estrutura do Web Services dopservice tende a ser extremamente simples. A seguir descreveremos a função de cada um dos módulos que o compõe e em seguida apresentaremos esquema ilustrando o relacionamento entre os mesmos.

- divulgaOrcaIF.Java – interface que expõe os serviços a serem oferecidos como parte do Web Services dopservice programado. Neste módulo constam apenas os nomes dos métodos e seus parâmetros de entrada e de retorno, bem como as exceções que estes podem produzir;

- divulgaOrcaImpl.Java – módulo de implementação de todos os métodos expostos pelo divulgaOrcaIF.Java. No caso desta demonstração temos apenas um método implementado nessa classe : msgxml (int) que recebe um inteiro e retorna uma String, como já mencionado. Essa classe instância a classe DopDAO2 para obter acesso ao banco de dados utilizando o método DopDAO2.posicaoOrca () que retorna o documento XML com a posição do orçamento;
- DopDAO2.Java – classe auxiliar que acessa o banco de dados no qual é executado o orçamento, como citado em [33], essa é uma classe de abstração de dados para o Web Services que está livre do contato físico com o banco de dados o qual pode ser trocado sem que haja a necessidade de alteração da classe de implementação. Outra característica importante desta camada de abstração é a utilização do JNDI para acessar o recurso fonte de dados (Data Source) disponível no container. Essa facilidade torna-se interessante pois temos o gerenciamento, via container, do pool de conexões com a fonte de dados ou seja o banco. Isto é um aperfeiçoamento que aproxima ainda mais o JWSDP da tecnologia J2EE, como pode ser notado no uso deste recurso em [57];

A Figura 6.3.3-1 relacionamento entre os módulos do Web Services dopservice, a seguir, demonstra como os módulos descritos se interrelacionam, bem como o posicionamento destes dentro do container Tomcat. Observe que os mesmos são denominados ainda com a extensão .class, pois é somente após compilados que devem ser carregados para dentro do container, o que não altera o relacionamento que desejamos demonstrar.

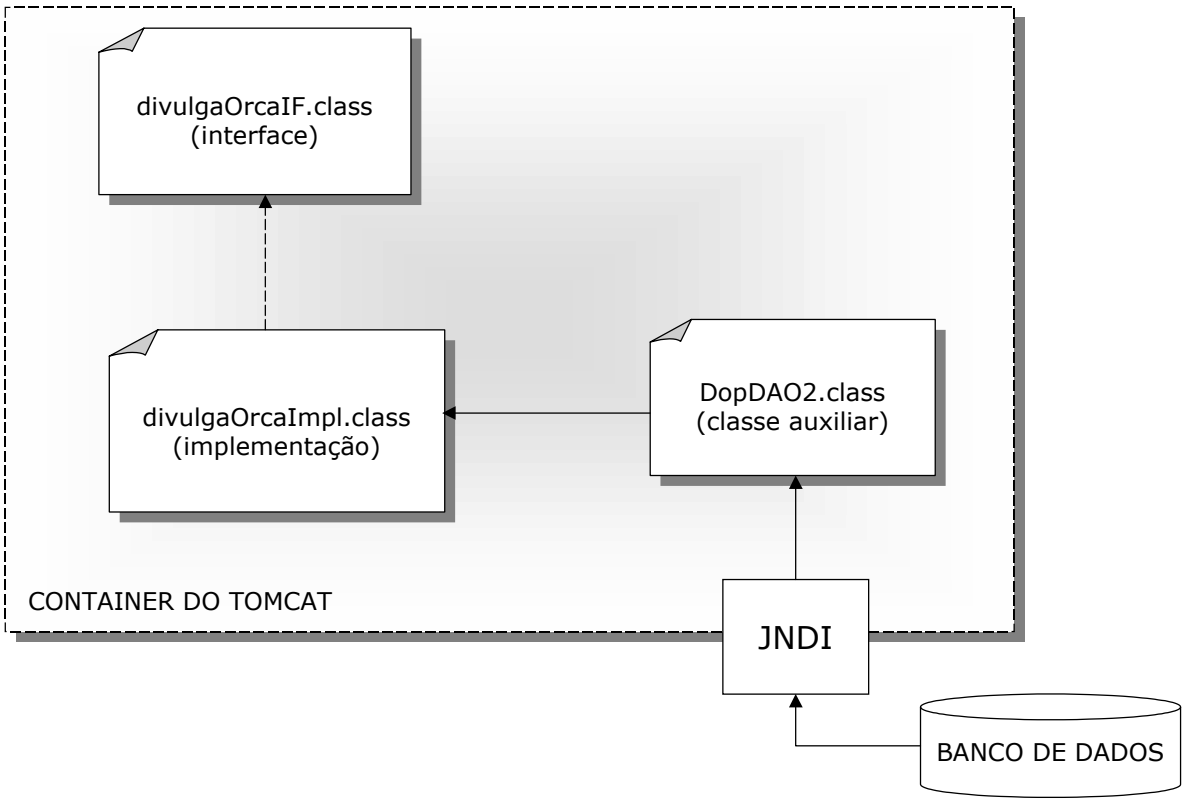


Fig. 6.3.3-1 relacionamento entre os módulos do Web Services dopservice.

Através do emprego dos utilitários wscompile e wsdeploy da forma como fizemos e demonstramos anteriormente, todo o trabalho de desenvolvimento de Web Services em Java constitui-se no projeto e programação cuidadosos dos serviços que serão oferecidos como parte do Web Services.

Conforme [10] e [11], os utilitários fornecidos com JWSP, na versão 1.3, wscompile e wsdeploy criam camadas de Servlets especiais e apropriadas para receber solicitações em HTTP e SOAP, processam as solicitações vindas no envelope SOAP e ativam as classes capazes de responde-las em Java, tomam a resposta, colocam-na no envelope SOAP e a retornam ao solicitante. Na Figura 6.3.3-2 abstração simplificada de Web Services implementados em Java, visão de como dopservice responde a solicitações.

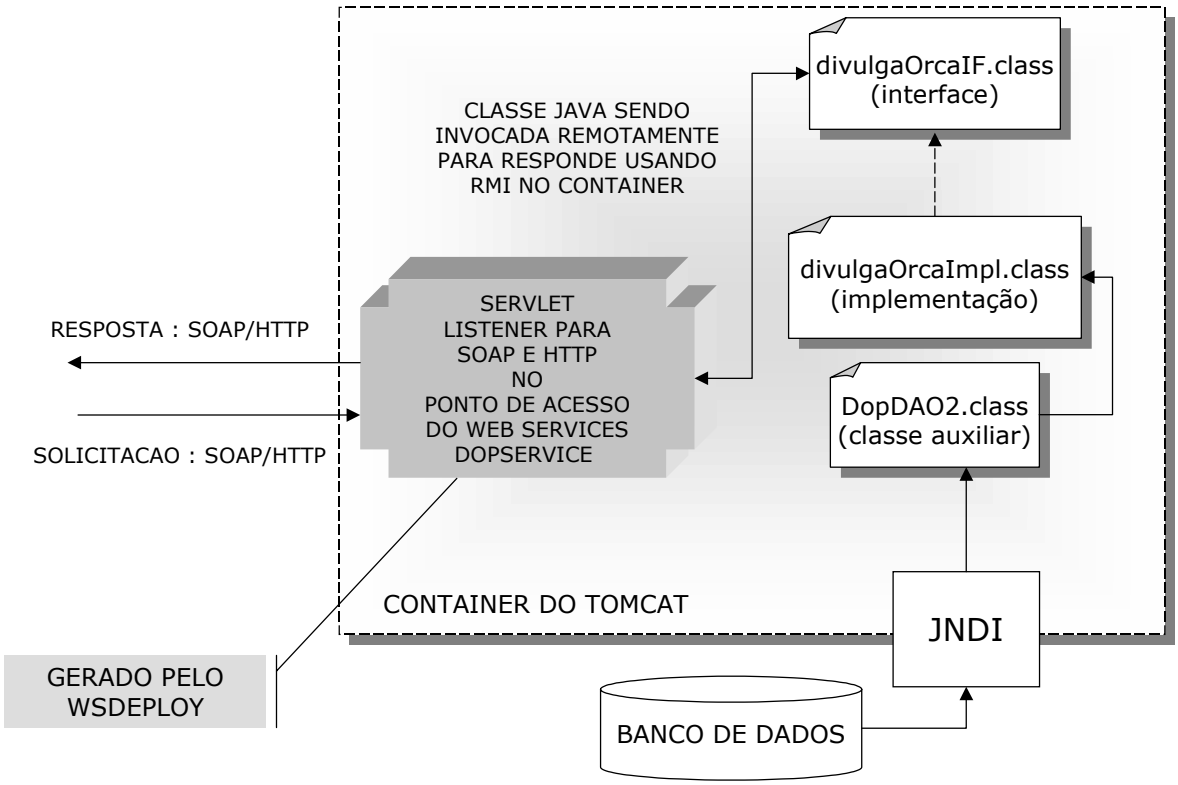
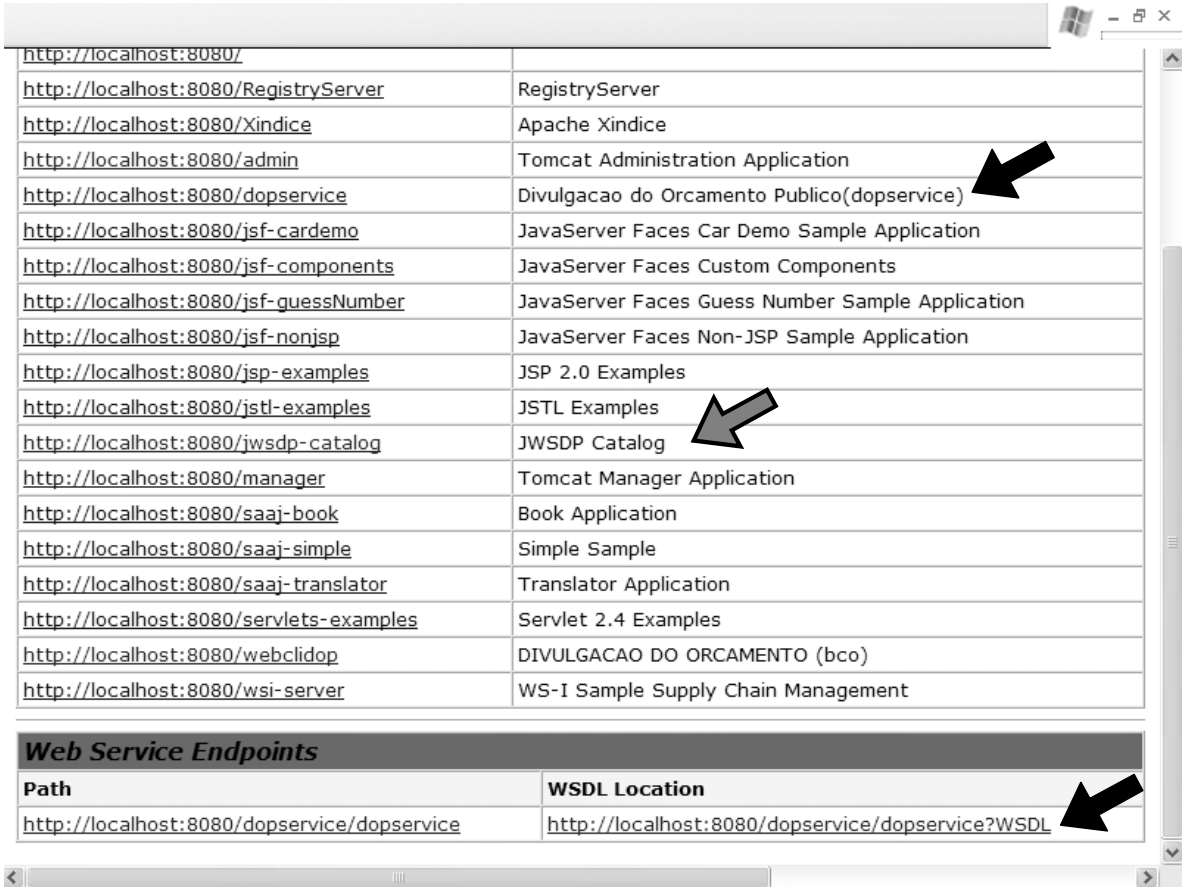


Fig. 6.3.3-2 abstração simplificada de Web Services implementados em Java.

6.3.4 – Telas e interface com o usuário

Por definição um Web Services deve apresentar apenas uma URL na qual ele poderá ser acessado e onde o mesmo deverá fornecer o WSDL que o descreve [12]. Em nossa demonstração temos três telas a apresentar : o Web Services dopservice publicado (deploy) no Tomcat, a tela de definição de ativo do ponto de acesso (endpoint) e a exposição de parte do WSDL do mesmo.

A publicação do Web Services dopservice no JWSDP, foi realizada através da função deploy presente no aplicativo Application Manager que constitui-se em interface gráfica com o usuário. Nesse caso, fornecemos o arquivo dopservice.war e o Web Services nele definido passou a ser disponibilizado no catálogo de aplicações do JWSDP e para consumo público. Podemos observar que a publicação (deploy) do Web Services foi realizada com sucesso, através da aplicação JWSDP Catalog, também disponível no pacote e de acesso público, para que os possíveis interessados em consumir Web Services disponíveis no site, possam ter acesso as suas principais características de modo rápido e simplificado. A Figura 6.3.4-1 Web Services dopservice publicado no JWSDP, a seguir, demonstra concretamente o descrito através de uma tela extraída de consulta feita ao aplicativo JWSDP Catalog, as setas tem o objetivo de ajudar a localização dos aplicativos citados.



http://localhost:8080/	
http://localhost:8080/RegistryServer	RegistryServer
http://localhost:8080/Xindice	Apache Xindice
http://localhost:8080/admin	Tomcat Administration Application
http://localhost:8080/dopservice	Divulgacao do Orcamento Publico(dopservice)
http://localhost:8080/jsf-cardemo	JavaServer Faces Car Demo Sample Application
http://localhost:8080/jsf-components	JavaServer Faces Custom Components
http://localhost:8080/jsf-guessNumber	JavaServer Faces Guess Number Sample Application
http://localhost:8080/jsf-nonjsp	JavaServer Faces Non-JSP Sample Application
http://localhost:8080/jsp-examples	JSP 2.0 Examples
http://localhost:8080/jstl-examples	JSTL Examples
http://localhost:8080/jwsdp-catalog	JWSDP Catalog
http://localhost:8080/manager	Tomcat Manager Application
http://localhost:8080/saaj-book	Book Application
http://localhost:8080/saaj-simple	Simple Sample
http://localhost:8080/saaj-translator	Translator Application
http://localhost:8080/servlets-examples	Servlet 2.4 Examples
http://localhost:8080/webclidop	DIVULGACAO DO ORCAMENTO (bco)
http://localhost:8080/wsi-server	WS-I Sample Supply Chain Management

Web Service Endpoints

Path	WSDL Location
http://localhost:8080/dopservice/dopservice	http://localhost:8080/dopservice/dopservice?WSDL

Fig. 6.3.4-1 Web Services dopservice publicado no JWSDP.

Estando o Web Services dopservice corretamente publicado, o próximo estágio de conferência do seu funcionamento e característica para uso é a consulta a sua tela de ativação que pode ser feita clicando-se na URL apresentada anteriormente ou apontando-se o browse para a mesma que no caso seria http://localhost:8080/dopservice/dopservice, ao acessar essa URL o servidor de Web Services JWSDP apresenta tela com três grupos de informações : o nome da porta de acesso ao Web Services, no caso DivulgaOrcamento, o Status da porta que pode ser em funcionamento (ACTIVE) ou parado (INACTIVE) e o terceiro e maior grupo de informações contendo o endereço do Web Services, já citado, a WSDL onde pode-se obter o respectivo arquivo, http://localhost:8080/dopservice/dopservice?WSDL, a porta Qname que representa a porta de acesso para o Web Services, uma espécie de nome que deve ser usado para ativar a utilização do mesmo em qualquer linguagem ou plataforma, conforme indicado em [11], nessa demonstração representado por {http://abilio.com/wsdl}divulgaOrcaIFPort, onde o conteúdo entre as chaves querem indicar apenas diferenciadores e prefixos do XML, denominados de Namespaces, como citado em [12] uma das vantagens do XML Schema. Como o Web Services foi implementado em Java e trata-se do ambiente Java, as classes de implementação que expõem o Web Services são citadas em Remote interface, com a interface dopservice.divulgaOrcaIF e Implementation class : dopservice.divulgaOrcaImpl.

A última informação desta tela constitui-se Model que indica a URL http://localhost:8080/dopservice/dopservice?model, nessa URL podemos baixar cópia do arquivo model.gz que segundo [63] constitui-se na descrição do Web Services para o serviço de interface do ponto de acesso (sigla s-e-i de Services endpoint interface) o qual constitui-se na parte que faz com que o Servlet especial receba as invocações Web Services (SOAP/HTTP) traduza em Java, passe a quem deve responde-la, aguarde a resposta, receba a resposta em Java e transforme a resposta em SOAP/HTTP e responda ao solicitante. A Figura 6.3.4-2 informações do JWSDP sobre o Web Services dopservice, a seguir demonstra cada um dos itens descritos anteriormente.

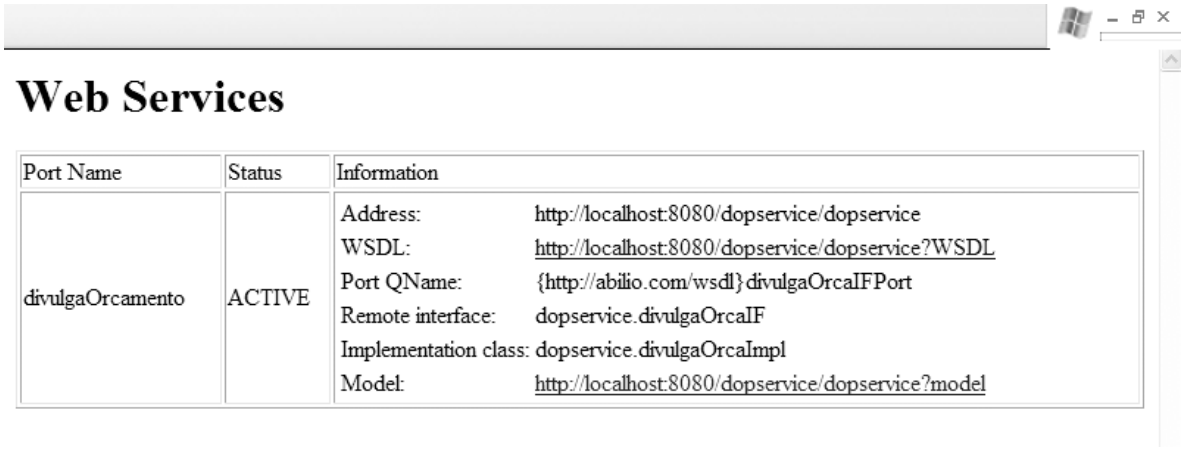


Fig. 6.3.4-2 informações do JWSDP sobre o Web Services dopservice.

Com a disponibilização do arquivo descritor do Web Services dopservice, os potenciais clientes para o seu consumo podem realizar a geração de suas interfaces de acesso e obter as informações e os serviços descritos. A obtenção do WSDL torna-se bastante simples, bastando apenas ao usuário do browser solicitar que o mesmo grave o arquivo com o nome e a extensão que desejar. A Figura 6.3.4-3 arquivo WSDL do Web Services dopservice mostrado pelo browser, a seguir, fornece exemplo do que ocorre quando este tipo de arquivo, um documento XML bem formado, é acessado e mostrado pelo browser.

Fig. 6.3.4-3 arquivo WSDL do Web Services dopservice mostrado pelo browser.

6.4 – Desenvolvimento do consumidor Web do Web Services dopservice

Conforme anteriormente indica na Seção 6.2.6, essa demonstração cobre a implementação de um consumidor do Web Services dopservice implementado. Seguindo o projeto iniciado no Capítulo 5, Seção 5.3.2, passamos a descrever o desenvolvimento do cliente para o referido Web Services. Chamamos este cliente de webclidop, acrônimo para a junção das palavras Web, porque deverá rodar sob browser, cli de cliente pois consumirá um Web Services e dop que é a abreviatura para divulgação do orçamento público, tema deste estudo de caso.

Para o desenvolvimento do webclidop, elaboramos aplicação em Java baseada nas tecnologias canônicas indicadas em [63] e anteriormente já estudadas em [57]. Combinamos a tecnologia de Servlet e JavaServer Pages, em padrões de programação adotando alguns dos aspectos de MVC (Model View Controler Modelo Visão Controle), conforme indicado em [20], [44].

Aproveitando sugestão encontrada em [10], empregamos aspectos avançados de JSP (JavaServer Pages), a partir de bibliotecas já desenvolvidas e prontas TLD (Tag Lib Descriptor,

bibliotecas de descrição de sinalizadores) no que diz respeito ao processamento e utilização de documentos XML. Nossa decisão teve como base dois motivos : demonstrar o poder do XML e aplicar, na prática, o reaproveitamento, a reusabilidade de componentes.

Assim, a construção do webclidop, teve como linha orientadora a simplicidade de sua interface com o usuário, adornada com o requinte de estar consumindo dados produzidos pelo Web Services dopservice e processando documentos XML com tecnologia JSP TLD e Xpath de XML.

6.4.1 – Requisitos do webclidop

Os requisitos aqui descritos, tratam de oferecer visão particular, destaque, sobre pontos e detalhes do desenvolvimento, que tornam essa aplicação, consumidora do Web Services dopservice, diferenciada de uma aplicação do mesmo tipo que realize função semelhante, porém sem entretanto aplicar as tecnologias aqui empregadas :

- o consumo dos dados apresentados pelo webclidop, ocorre a partir do Web Services dopservice, que poderia ser desenvolvido em qualquer plataforma. Este consumo ocorre através de classes geradas (a partir do WSDL publicado do Web Services) que funcionam como camada de acesso ao Web Services, e fornecem os dados requeridos através de métodos de classes Java. A seguir apresentamos parte do código da classe DocXmlWS, na qual a conexão e o acesso ao Web Services dopservice é realizado. Observe que a partir da criação do objeto para acesso a porta do Web Services e da configuração do ponto de acesso, é criado uma instância da classe DivulgaOrcaIF_Stub (gerada a partir do WSDL fornecido pelo dopservice) que permitirá acesso normal a todos os serviços disponíveis, por exemplo wserv.docxml () retornará o String, com o documento XML bem formado contendo a posição da execução orçamentária, conforme descrito na Seção 6.3 :

```
Stub stub = (Stub)
            (new DivulgaOrcamento_Impl ().getDivulgaOrcaIFPort ());

stub._setProperty
    (Javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,endpointAddress);

wserv =(DivulgaOrcaIF_Stub) stub;
```

- a String contendo documento bem formado em XML composto de todas as informações sobre a execução do orçamento é passada para o contexto da aplicação, o seu processamento ocorre de

modo simplificado, com o apoio de TLD, no módulo JavaServer Pages que fornece a informação projetada como resultado final ilustrado na Seção 5.3.2. A seguir parte do código JSP do módulo posicaoOrca.jsp, que processa o documento XML enviado pelo Web Services dopservice sendo armzenado na ${xml}.

```
<%@ taglib prefix="x" uri="http://Java.sun.com/jstl/xml" %>

<x:parse xml="${xml}" var="tabelaorca" scope="application" />

                  .
                  .
                  .
    (instruções HTML que tornam a interface com o usuário atraente)
                  .
                  .
                  .

<x:forEach select="$tabelaorca//linha">

        <%
          if  (flag)
           { cor_fundo = "c090ff";
             flag = false; }
           else
           { cor_fundo = "c0c0ff";
              flag = true; }
        %>

        <tr bgcolor="<%=cor_fundo%>" >
          <td>
            <div align=left>
               <x:out select="./orgao"/>
            </div>
          </td>


          <td align=right>
              <x:out select="./orcado"/>
          </td>

          <td align=right>
              <x:out select="./pago"/>
          </td>

        </tr>

</x:forEach>
```

Considerando os pontos destacados anteriormente, os quais possibilitam a verificação prática da aderência da tecnologia Java ao emprego efetivo de Web Services e das facilidades e do poder que a linguagem XML agrega ao todo. Tornando este modelo de computação distribuída, como citado

em [10] e [11] uma opção concreta de adoção para a comunicação entre programas que desejem utilizar a Internet para comunicar-se e compartilhar recursos.

6.4.2 – Fases do desenvolvimento do webclidop

De posse do WSDL fornecido pelo Web Services dopservice e com o projeto do webclidop definido para atender ao que fora especificado na Seção 5.3.2, passamos a construção do mesmo partindo dos dados a serem obtidos do Web Services. A seguir pontuamos cada uma das etapas percorridas :

- obtenção do WSDL do Web Services dopservice para a geração das classes de acesso ao mesmo, também chamadas de classes Stub;
- com o emprego do utilitário wscompile, fornecido pelo Java WSDP 1.3, simplesmente fornecemos a este utilitário o arquivo WSDL do Web Services dopservice o obtemos todas as classes em Java para acesso ao mesmo. Pode-se ainda obter os fontes dessas classes, se for necessário;
- desenvolvemos as páginas em JSP e um Servlet de controle, obedecendo aos princípios, já citados, do MVC;
- seguindo orientação de [57] e principalmente de [33] , isolamos o acesso a fonte de dados, no caso o Web Services dopservice em uma classe Java separada que acessa o mesmo e passa as informações para o Servlet de controle.
- empregamos ainda o método init (Servlet config) para criar uma instância da classe de acesso ao Web Services e emulamos cache, seguindo o recomendado em [55].

Como pode ser observado, a partir da produção das classes Stub de acesso ao Web Services desejado, todo o trabalho de desenvolvimento em Java, para a utilização dos serviços oferecidos pelo Web Services, constitui-se no cuidadoso emprego de conceitos já bastante disseminados e estabelecidos entre os desenvolvedores desta tecnologia, como preconiza [55] de forma subliminar. A Figura 6.4.2-1 geração de classes Stub para acesso ao Web Services dopservice a partir do WSDL, ilustra como estas classes podem ser obtidas a partir do arquivo WSDL fornecido, que constitui-se em um documento XML válido e bem formado

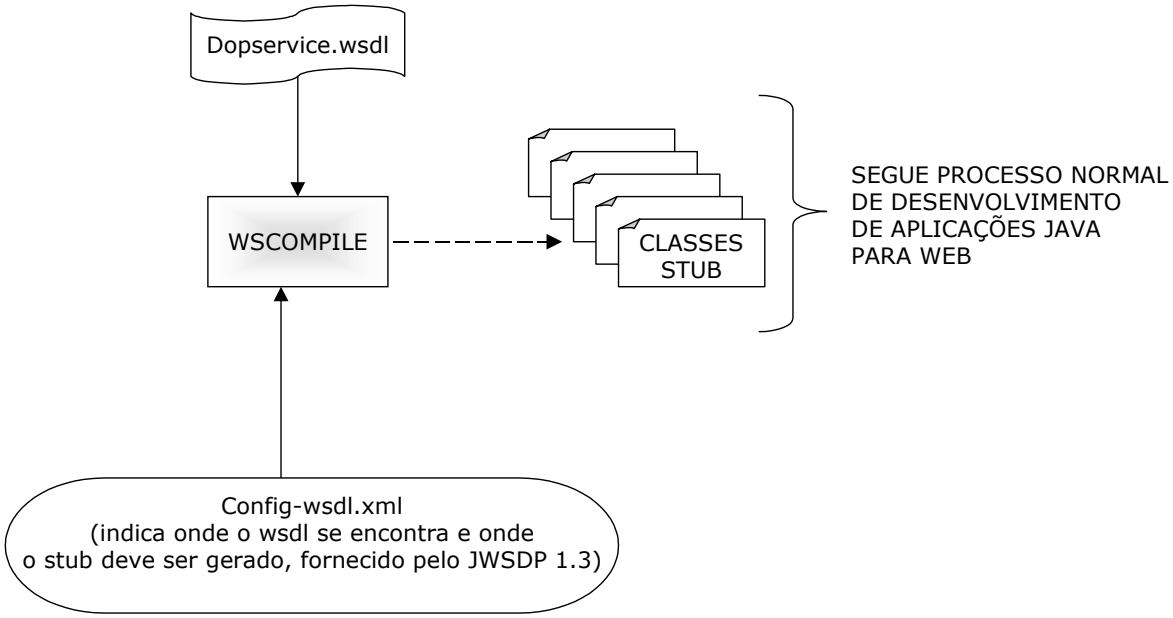


Fig. 6.4.2-1 geração de classes Stub para acesso ao Web Services dopservice a partir do WSDL.

6.4.3 – Estrutura do software e módulos

Com a adoção da arquitetura MVC e em se tratando de uma aplicação que tem como objetivo possibilitar o emprego da tecnologia de Web Services e um pouco de XML avançado para extrair e exibir informações sobre a execução orçamentária, os módulos principais que compõem a aplicação consumidora do Web Services dopservice, denominada webclidop serão descritos a seguir :

- index.jsp – módulo de abertura, tela inicial que permite a escolha de três opções : Posição do Orçamento por Órgão, Posição do Orçamento em Documento XML e a descrição, o resumo sobre a demonstração, que trata-se de uma tela explicando como a demonstração funciona. Este módulo JSP informa ao Servlet de controle qual a opção selecionada;
- SvrControle.Java – servlet de controle, responsável por gerenciar a navegação entre as páginas e por criar o acesso e a recuperação aos dados do Web Services dopservice, através da instanciação da classe DocXmlWS que fornece o String obtido a partir do Web Services. Neste módulo criamos um atributo de contexto com o conteúdo retornado do Web Services que simula cache de forma simplificada. Este Servlet controla o acesso as duas opções que utilizam a informação proveniente do Web Services : Posição do Orçamento por Orgão, Posição do Orçamento em Documento XML;
- posicaoOrca.jsp – módulo que toma o atributo de memória mantido e atualizado pelo SvrControle, com o conteúdo de um documento bem formado XML e transforma este dado em informação planejada na Seção 5.3.2, empregando a TLD x e a tecnologia Xpath do XML. Este módulo

representa concretamente a aplicação e a funcionalidade da tecnologia Web Services, no contexto geral deste trabalho;

- posicaoXML.jsp – toma o documento XML bem formado mantido e atualizado no contexto da aplicação pelo servlet de controle SvrControle e através do seu processamento adequado, o apresenta para o usuário. A particularidade deste módulo está em fazer com que o browser ative seus mecanismos de exibição para XML como se estivesse abrindo um arquivo com essa extensão e mostrando o seu conteúdo da forma padronizada e organizada. Para conseguir tal intento tivemos que construir um pequeno processamento em JSP para o conteúdo do documento válido XML armazenado na variável de contexto.
- DocXmlWS.Java – módulo que trata da abstração da fonte de dados. Este módulo ativa o Web Services dopservice, cria uma instância do objeto fornecido pelo mesmo denominado divulgaOrcaIf_Stub e a partir deste objeto fornece ao servlet de controle, os serviços disponibilizados, no caso através do método DocXmlWS.posicaoOrca ();

Esses são os principais e mais relevantes módulos que compõe a aplicação, consumidora do Web Services doservice, denominada webclidop. Entretanto existem outros módulos que são empregados para contribuir no efeito estético da aplicação como por exemplo a página de erro que é carregada sempre que ocorrerem erros em qualquer das páginas de extensão .jsp, existem ainda páginas de estruturação e de apresentação de cabeçalhos, todas utilizadas para tornar o aspecto estético da aplicação mais agradável ao usuário. A seguir a Figura 6.4.3-1 relacionamento dos módulos que compõe a aplicação webclidop no container Tomcat, mostra como estes módulos relacionam-se entre si. Dois detalhes a observar : os módulos em Java apresentam a extensão .class e as setas indicam apenas relações primárias de dependência e não possibilidades ou mapeamento de navegação absoluta entre os mesmos.

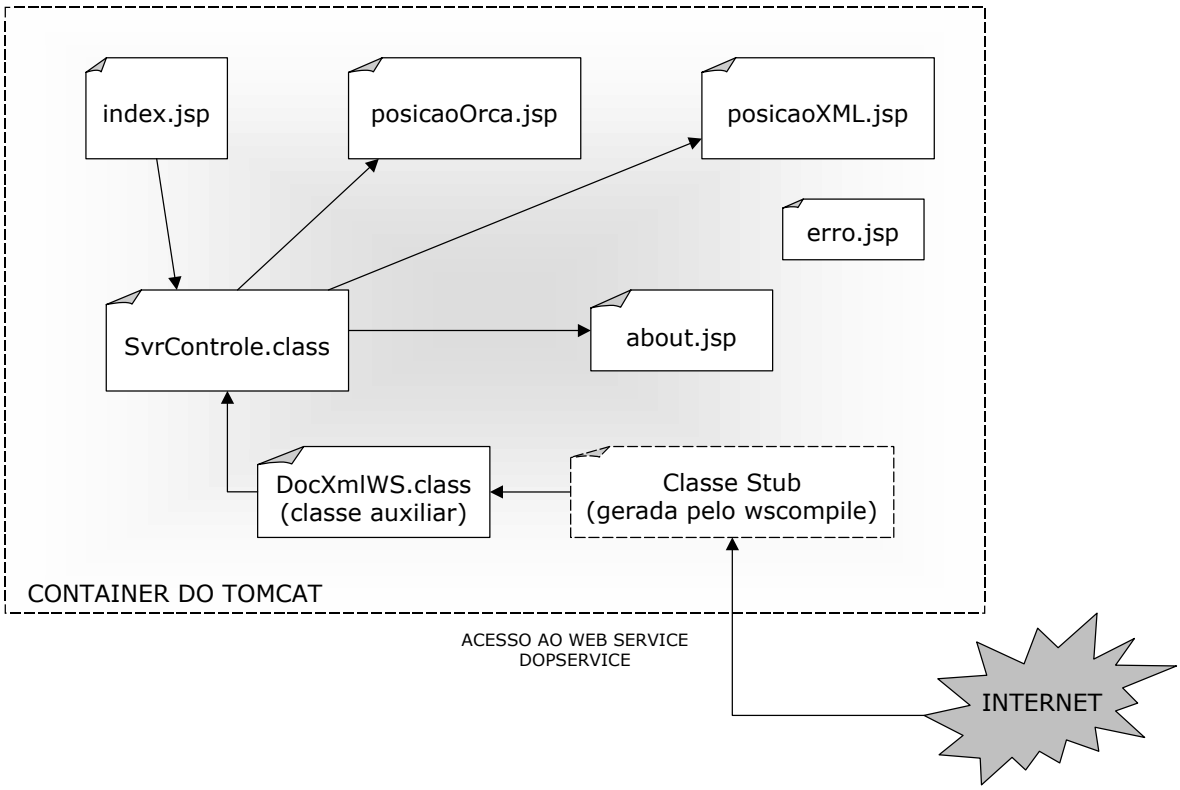


Fig. 6.4.3-1 relacionamento dos módulos que compõe a aplicação webclidop no container Tomcat.

6.4.4 – Telas e interface com o usuário

A interação da aplicação webclidop com o usuário se dá somente através de três telas realmente importantes e duas que poderão ou não ser visitadas. A primeira tela com a qual o usuário terá contato é a tela de abertura que apresenta as opções para as quais ele poderá navegar, são ao todo três : Posição do Orçamento Público por Órgão, Posição do Orçamento em Documento XML e Sobre este estudo de caso, que oferece um pequeno resumo sobre a aplicação. A Figura 6.4.4-1 tela de abertura da aplicação webclidop, oferce uma visão desta tela.

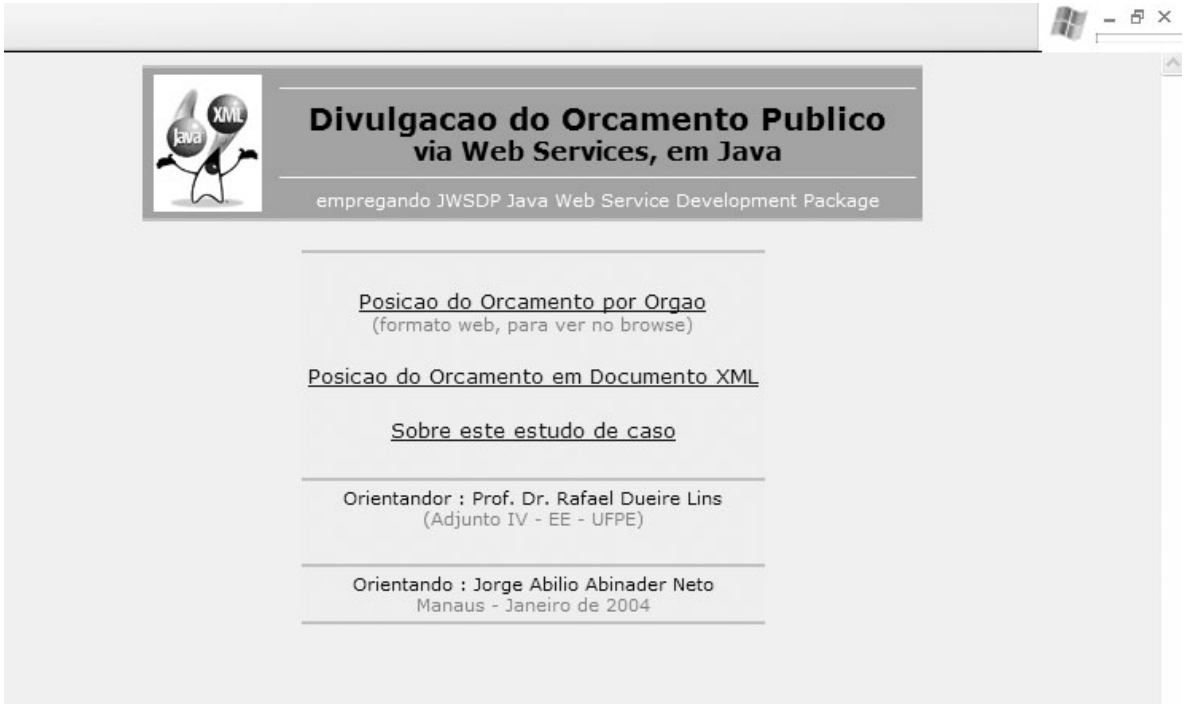


Fig. 6.4.4-1 tela de abertura da aplicação webclidop.

Ao escolher a opção Posição do Orçamento por Órgão, a primeira, o usuário terá acesso a tabela contendo o nome da cada órgão, quanto foi orçado para ser investido no ano corrente por aquele órgão e uma terceira coluna, indicando quanto foi efetivamente investido, a Figura 6.4.4-2 tela da opção Posição do Orçamento Público por Órgão, ilustra a mesma a seguir.

Posicao do Orcamento Publico
por Orgao

Data e Hora : Wed Jan 07 19:16:28 GMT-04:00 2004 (deste computador)

Retorna a pagina inicial

ORGAO	VALOR ORCADO	VALOR EMPREGADO
SECRETARIA MUNICIPAL DE EDUCACAO	131,836,036.00	44,446,496.96
SECRETARIA MUNICIPAL DE OBRAS, SANEAMENTO BASICO E SERVICOS PUBLICOS	113,483,349.00	63,112,177.32
SECRETARIA MUNICIPAL DE SAUDE	74,296,878.00	37,678,625.85
RECURSOS SUPERVISIONADOS PELA SEMEF	27,540,725.00	18,834,951.48
FUNDO MUNICIPAL DE SAUDE	25,143,479.00	4,315,523.32
CAMARA MUNICIPAL DE MANAUS	24,400,000.00	5,776,416.98
SECRETARIA MUNICIPAL DE ECONOMIA E FINANCAS	18,850,000.00	9,581,406.03
INSTITUTO MUNICIPAL DE PREVIDENCIA E ASSISTENCIA SOCIAL	15,987,000.00	
RECURSOS SUPERVISIONADOS PELA SEMAD	15,885,000.00	8,053,847.13
SECRETARIA MUNICIPAL DE ASSISTENCIA SOCIAL E CIDADANIA	10,633,636.60	4,309,338.96
SECRETARIA MUNICIPAL DE ABASTECIMENTO, MERCADOS E FEIRAS	8,610,000.00	4,011,565.48
GABINETE CIVIL	6,835,000.00	2,907,327.32
SECRETARIA MUNICIPAL DE ADMINISTRACAO	6,825,000.00	3,672,782.53
RECURSOS SUPERVISIONADOS PELO GABINETE CIVIL	6,309,275.00	3,131,742.62

Fig. 6.4.4-2 tela da opção Posição do Orçamento Público por Órgão.

Para demonstrar como realmente o Web Services dopservice envia, dentro de String, documento bem formado XML, construímos e disponibilizamos a opção Posição do Orçamento em Documento XML. Ao ativar essa opção o usuário terá conhecimento da quantidade de dados constantes no documento XML e de que somente um subconjunto destes está sendo usado, podendo-se expandir ainda mais. Temos uma pequena amostra de parte do documento na Figura 6.4.4-3 tela da opção Posição do Orçamento em Documento XML. Observe que cada linha corresponde a um órgão e tem como elementos de sua composição outros elementos não mostrados.

Fig. 6.4.4-3 tela da opção Posição do Orçamento em Documento XML.

Como optamos pela tecnologia JSP, devemos ter a previsão de que erros e exceções imprevistos pode ocorrer com as demais páginas desta tecnologia. Assim, montamos a página de erro que poderá ser ativada caso ocorra algum mal funcionamento, como por exemplo a ausência do Web Services dopservice que pode ocorrer por qualquer motivo. A Figura 6.4.4-4 página de erro para a camada JSP, demonstra a página em funcionamento durante um erro simulado com a desativação do banco de dados da execução orçamentária.

Fig. 6.4.4-4 página de erro para a camada JSP.

6.5 – Limitações

O desenvolvimento do Web Services dopservice e da aplicação Web para consumo das informações oferecidas por este, possibilitou estudo da tecnologia Java para desenvolvimento de Web Services. Além da capacitação adquirida e da constatação de alguns aspectos teóricos empregados na prática, temos a destacar, nos pontos a seguir, conjunto reduzido de limitações que consideramos relevantes para o desenvolvimento do objetivo de popularizar a tecnologia Java na divulgação do orçamento público via Web Services.

- o Java WSDP 1.3, versão utilizada, apresenta a possibilidade da implementação de clientes em três categorias de funcionamento diferentes, especificadas pela tecnologia Java para Web Services : Stub, Proxy e DII. Implementamos a versão Stub, por tratar-se do modo mais indicado para aplicações reais, segundo o contexto deste estudo de caso;
- mesmo com toda a documentação disponível, existem ainda grandes quantidades de detalhes a serem esclarecidos, na tecnologia Java, com relação a sua aderência a Web Services, como por exemplo o fato de os utilitários wscompile e wsdeploy, estarem disponíveis em documentação separada e não constarem de modo claro no documento tutorial, como pode ser observado em [63] ;

- a adaptação do container Tomcat para que Java tenha aderência a tecnologia Web Services, demonstra constituir-se uma opção que tende a tornar-se estável, porém a curva de aprendizado e o domínio pleno da combinação do container Tomcat e Web Services em Java, requer tempo e empenho, diferente da realidade que tentam passar fabricantes como em [55];
- nessa demonstração, seguimos a sugestão encontrada em [55] e ignoramos a necessidade de pesquisar o Web Services desejado em um UDDI. O estudo profundo deste assunto revela que a inserção de um Web Services de forma correta, segundo o padrão que tenta estabelecer a API JAX R, conforme [10] e [11], torna-se tarefa destinada a especialistas no ramo de negócios em que se deseja disponibilizar o Web Services. Pode ocorrer de o detalhamento e a exposição do Web Services no UDDI escolhido tornar-se mais complexo que sua implementação e disponibilização. Diante desta possibilidade concreta, decidimos por implementar um Web Services e a partir deste o seu consumidor, que já nasce com o conhecimento do que vai consumir e empregar;
- a demonstração apresenta o funcionamento adequado em ambiente controlado, porém não houve oportunidade para testes maiores, incluindo ambiente real da rede metropolitana, acesso ao Web Services de dentro desta rede e também de fora.
- seria interessante a realização de testes de acesso do Web Services dopservice através de outras linguagens e plataformas. Temos em [10] um exemplo citando o acesso via .NET [32] .

Todas estas limitações, constituem-se fatos pertinentes a tecnologia escolhida e tornam a demonstração apresentada válida como estudo inicial de aplicação da tecnologia Java no cenário definido.

Entretanto, a implementação concreta, real e destinada ao consumo industrial de Web Services a serem disponibilizados para uma comunidade, em qualquer esfera de governo, necessitará de maior aprofundamento diante das limitações anteriormente citadas e investimentos em software, hardware e pessoal, os quais deverão estar dimensionados para atender a demanda estimada.

Comprovamos com nossa demonstração a viabilidade do emprego de tecnologia Java para a divulgação do orçamento público via Web Services, sua expansão e utilização prática constitui-se a continuação do caminho por nós iniciado.

6.6 – Conclusão

Demonstramos, a partir do pacote Java WSDP, fornecido gratuitamente pela Sun MicroSystems, os passos percorridos para o desenvolvimento e publicação do produtor de Web Services dopservice e da aplicação consumidora deste serviço, o cliente webclidop. Ambos construídos em tecnologia Java. Vale ressaltar que para essa tecnologia faz-se necessário a instalação do pacote Java WSDP tanto no fornecedor do Web Services quanto nos consumidores do mesmo. Analisando mais detalhadamente este requisito, somos levados a concluir que a tecnologia Java, fazendo uso do Java WSDP, adere aos padrões da Internet, habilitando-se integrar "fontes de dados" disponibilizadas via Web Services às suas aplicações distribuídas.

Além da implementação, oferecemos prova de conceito que garante a viabilidade da implementação prática para divulgação do orçamento público via Web Services em Java. Respeitando-se as limitações verificadas no desenvolvimento do protótipo.

Na parte seguinte, concluiremos este estudo de caso discorrendo nossas considerações finais, resumindo as principais contribuições deste trabalho sobre a popularização do uso de Web Services em Java, para divulgação do orçamento público e sugerindo trabalhos futuros que podem ter como base inicial o conteúdo deste.

CAPÍTULO 7 – Conclusões

7.1 – Considerações finais

O surgimento de grandes redes de computadores e o natural aparecimento da Internet, que interligando essas grandes redes tornou-se a maior e mais complexa de todas, expandiu o meio ambiente anterior dos sistemas de computação distribuída. Na tentativa de interligar aplicações e possibilitar a troca automatizada de dados entre sistemas aplicativos, surgiram alguns modelos importantes para suportar aplicações distribuídas no ambiente de Internet. Cada um desses modelos estabelece requisitos e características próprias que devem ser adotados e repetidos por aqueles que desejam comunicar-se través dos mesmos, no ambiente de Internet estabelecido.

O ambiente de Internet possibilita amplo espaço para pesquisa, destacando-se como alternativa para computação distribuída e comunicação efetiva entre sistemas aplicativos. Nessa perspectiva, a tecnologia Web Services apresenta-se como uma das alternativas viáveis. Considerando-se aspectos fundamentais de sua natureza como : baixo acoplamento entre as partes; independência de plataforma e linguagem e a inexistência da adição de requisitos estranhos ao ambiente de Internet[3].

Baseada em arquitetura orientada a serviços, a tecnologia de Web Services possui como estrutura básica e núcleo, três atores (construídos sobre XML) com papéis bem definidos : SOAP que constitui-se no protocolo de comunicação, sem requisitos sobre a camada inferior; WSDL, linguagem de descrição do Web Services e UDDI diretório para a publicação, busca e descoberta de Web Services disponíveis que reúne a oferta desses e torna a sua localização simplificada.

A combinação dos entes SOAP, WSDL e UDDI, torna a tecnologia de Web Services única, por conferir-lhe caráter contratual, de modo que ao encontrar o Web Services desejado e tomar conhecimento das exigências para seu consumo, o potencial cliente/consumidor, em qualquer

linguagem ou plataforma, poderá empregá-lo da forma que considerar mais adequada. Sem a necessidade de intervenção humana.

No que se refere a tecnologia Java, considera o surgimento de Web Services como uma possibilidade de expansão de clientes em potencial que poderão utilizar os benefícios da computação distribuída via Internet, como enfatizam [55] e [9]. Mesmo oferecendo o modelo de implementação para sistemas distribuídos J2EE [6],[7], a tecnologia Java estende-se para absorver e compatibilizar-se com os três elementos principais que constituem a tecnologia de Web Services.

Através do conjunto de APIs da família JAX, Java disponibiliza em seu software para referência de implementação Java WSDP, APIs que permitem a implementação, disponibilização (deploy) e o consumo de Web Services criados a partir de programação Java [54].

De modo a reproduzir, em escala controlada, características de heterogeneidade e adversidades encontradas na Internet, selecionamos como cenário para nosso estudo de caso as redes metropolitanas brasileiras nas esferas municipal, estadual e federal. Assim, o contexto escolhido constituí-se ideal para a implementação e a disponibilização de Web Services que permitam a troca dinâmica de informações entre os órgãos de uma mesma esfera de governo.

O objeto sobre o qual atuou nosso estudo de caso, a execução do orçamento público, por operar com recursos financeiros [62], transforma-se em ponto de interesse que dispensa maiores justificativas.

A divulgação pública da execução orçamentária via Web Services em Java, constitui-se, como demonstrado, estudo de caso validado pela implementação do protótipo e completamente factível, o qual poderá ser adotado e executado em termos práticos e para funcionamento industrial, mediante a extensão de testes de escalabilidade e diversificação dos modos de acesso.

Nosso objetivo foi duplamente atingido, pois o modelo de disponibilização de informações via Web Services atende completamente a exigência principal do baixo acoplamento com o adicional do auto serviço (via disponibilização do arquivo WSDL). No aspecto da integração entre órgãos, o emprego de Java apresenta-se como opção concreta para a implementação de Web Services no cenário proposto, considerando-se a extrema facilidade e o amadurecimento reconhecido desta linguagem para conectar-se com uma infinidade de bancos de dados, como demonstra [24] e sua capacidade de disponibilizar e implementar conectores para sistemas legados, conforme indicado em

[57]. A implementação do protótipo demonstra ainda que Java pode vir a constituir-se em camada de exposição das informações dos órgãos para acesso e intercâmbio com o exterior.

7.2 - Contribuições

Entre os pontos relevantes que podem ser empregados como referência, informação introdutória ou complementar para os assuntos aqui abordados, este trabalho apresenta as seguintes contribuições :

- **estudo e revisão do estado da arte em Web Services implementados em Java**

Este estudo pode ser empregado como ponto de partida, fonte de consulta e embasamento para trabalhos que abordem a mesma linha de pesquisa na implementação de consumo e fornecimento de Web Services em Java. Apresentamos as APIs da família JAX : JAX P, JAX RPC, JAX M e JAX R associando suas funções com os elementos base, do núcleo, que constituem a tecnologia de Web Services e como se integram e interagem com os mesmos. Oferecemos discussão detalhada sobre a fundamentação e o funcionamento dos principais elementos desta tecnologia;

- **estudo e revisão dos principais modelos de computação distribuída para a Internet**

O emprego da Internet como meio de comunicação entre sistemas aplicativos, fez surgir, com o passar dos anos e o aumento da complexidade desse ambiente, várias propostas de soluções para a computação distribuída e a comunicação automática entre sistemas através desse ambiente de rede, considerado hostil e instável [3]. Abordamos de modo amplo os vários modelos existentes partindo dos mais antigos que eram utilizados em grandes redes locais e tentaram adaptar-se a Internet, passando por proposições da própria tecnologia Java e indo até soluções como a proposta J2EE [34] ;

- **desenvolvimento do protótipo de Web Service dopservice em Java**

Ao desenvolver e instalar o Web Service dopservice, disponibilizamos contribuição concreta para o desenvolvimento de novos Web Services de comportamento síncrono e a expansão desse. Nosso relato pode ser utilizado como mini tutorial de desenvolvimento ou documento inicial

para estudo de viabilidade de implementações similares. A utilização dos utilitários wscompile, wsdeploy, a configuração do Web Service, sua adaptação ao container do Tomcat [45] , sua ligação com o banco de dados Oracle [21], a utilização do recurso JNDI [50], os procedimentos relativos ao ciclo de projeto, implementação, publicação (deploy) e cofiguração do mesmo, todos estes detalhes estão disponíveis e reunidos em um único lugar, e podem contribuir para reduzir o caminho dos que decidirem seguir linha semelhante;

- **desenvolvimento do protótipo cliente consumidor do Web Service Webclidop em Java**

O cliente consumidor de Web Service Webclidop, tem como característica, simplicidade e sofisticação. Outros clientes podem ser desenvolvidos na mesma linha a partir do protótipo fornecido, que pode ser expandido e utilizado como referência de manipulação de documentos XML e consumo de Web Services;

- **execução do orçamento público**

Nesse aspecto oferecemos visão ampla, porém focada em pontos relevantes, da administração pública do orçamento, a qual pode ser empregada para qualquer uma das três esferas do poder do executivo. Nossa descrição da execução orçamentária, baseada em observações e constatações concretas, podem ser empregadas como ponto de partida inicial para trabalhos que tenham objetivo compreender este contexto para oferecer ao mesmo algum tipo de melhoria ou avanço tecnológico.

7.3 – Trabalhos futuros

Como extensão desse trabalho, temos a sugerir em trabalhos futuros os seguintes tópicos que podem ser desenvolvidos em complemento a linha de estudo de caso aqui iniciada :

- aplicação de Web Services assíncronos, utilizando-se a API JAX M na execução do orçamento público;
- comparação entre os ambientes Java WSDP e AXIS (Apache fundation), considerando aspectos de compatibilidade, complexidade de implementação, performance e segurança;

- análise comparativa de clientes consumidores de Web Services em Java, aplicações isoladas com interface gráfica para usuário, acessando Web Services de dentro do perímetro seguro da rede metropolitana e fora desse perímetro;
- análise comparativa de clientes consumidores de Web Services em outras linguagens e plataformas para acesso ao Web Service dopservice, com acesso de dentro do perímetro seguro da rede metropolitana e fora desse perímetro;

Os tópicos acima relacionados visam a melhoria e a expansão do tema desse estudo de caso. Nossa intenção com tais sugestões é possibilitar o aumento da massa de conhecimento sobre a aplicação de Web Services que podem ser usados de forma fluente na integração entre os órgãos públicos e na divulgação de informações que como prega a tecnologia de Web Services poderão ser consumidas por qualquer computador ligado a Internet, em qualquer linguagem e em qualquer plataforma.

Além do presente trabalho, a tecnologia de Web Services e sua relação com a linguagem Java permite espaço para pesquisa entre os quais selecionamos os que consideramos relevantes durante a realização desse estudo. Como por exemplo citado em [4] e considerado um dos principais problemas para uso pleno de Web Services, são os mecanismos de localização automática de Web Services dentro dos UDDI e usá-los de maneira correta. Outro tema de pesquisa citada na mesma fonte é a possibilidade de formalizar o modo de como compor um Web Services a apartir de outros Web Services, tarefa complexa, também chamada de Service Composition.

Uma das visões que podem ser adotadas como padrão, segundo sugere [4], é a de que Web Services devem ser concebidos como componentes que "moram" na Web e a partir desses deve-se poder construir aplicações. Entretanto, esta visão abre discussão para outros fatores a serem considerados como por exemplo disponibilidade, confiabilidade, no-repudiation, etc.

Apesar o avanço da tecnologia de Web Services e sua aceitação pela indústria da computação em [1], temos a ressalva sobre o protocolo HTTP que apresenta duas fortes limitações : não suporta acessos de longa duração e não consegue notificar eventos, a adaptação de SOAP para protocolos que atendam a estes requisitos pode ser tema de estudos futuros.

Para [1] a evolução de Web Services se dará inicialmente mantendo-se o baixo acoplamento, a base XML, os objetos podem ser usados para implementação de Web Services, mas não serão o modelo central e único de programação e a evolução será contínua através de Web Services Simples, protocolos alternativos e serviços de alto nível. A superação das limitações do HTTP e extensão das linhas de horizonte apontadas, podem constituir-se temas para pesquisas futuras.

Existe ainda, segundo [3], a possibilidade de no futuro aplicações virem a ser dinamicamente construídas a partir de Web Services, que serão dinamicamente selecionados no tempo de execução, a partir de critérios baseados em custo, qualidade, confiabilidade e disponibilidade.

Na utilização da interface JAX M, percebemos uma linha de pesquisa que pode ser seguida no sentido de solucionar o problema da correlação de mensagens assíncronas. Quando o solicitante envia o pedido, não há maneira formal na API JAX M para que a resposta, recebida algum tempo depois (horas, dias, semanas), seja associada a mensagem de solicitação, esta tarefa fica a cargo da camada do provedor, sobre o qual JAX M atua. O trabalho de pesquisa seria propor solução para este problema, talvez usando o mecanismo de Header do protocolo SOAP ou algum outro mecanismo.

8.0 - REFERÊNCIAS BLIBLIOGRÁFICAS

[1] EDWALD, Tim; "Undesrtanding XML Web Services The Web Services Idea"; Microsoft Corporation, 27-09-2002; Disponível em http://msdn.microsoft.com/webservices/undertanding/readme/default.aspx; Acessado em 05-05-2003.

[2] MANES, A Thomas; "Web Services Basics" ; Addison Wesley Professional; 15-08-2003; Disponível em http://www.informit.com ; Home>Articles>Web Development>Web Services, Acessado em 02-09-2003.

[3] CHAUDHARY, A. S., Saleem, M. A., Bukhari, H. Z.; "Web Services in Distribuited Applications Advantages and Problems"; Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Dsitrict Swabi, NWFP, Pakistan; Disponível em http://citeseer.nj.nec.com/548303.html, Acessado em 15-05-2003.

[4] CUBERA, F.; Nagy, W. A.; Weerawarana, S.; "Web Services : Why and How"; IBM T. J. Watson Research Center; 09-08-2001; Disponível em http://citeseer.nj.nec.com/488329.html, Acessado em 11-05-2003.

[5] Software AG, "WEB SERVICES Making the most of your business assets", Software AG, Março/2002.

[6] MONSON-HAEFEL, Richard; "EJB 2.1 Web Services: Part 1"; The ServerSide.com; Agosto/2002; Disponível em http://www.theserverside.com/resources/articles/MonsonHaefel-Column2/article.html; Acessado em 05-05-2003.

[7] MONSON-HAEFEL, Richard; "EJB 2.1 Web Services: Part 2"; The ServerSide.com; Setembro/2002; Disponível em http://www.theserverside.com/resources/articles/MonsonHaefel-Column3/article.html; Acessado em 26-06-2003.

[8] Systinet; "Web Services and J2EE"; The ServerSide.com; Fevereiro/2002; Disponível em http://www.theserverside.com/resources/articles/Systinet-web-services-part4/article.html; Acessado em 08-05-2003.

[9] SAGANICH, Al; "JSR 109 : Web Services Inside of J2EE Apps"; Ther O'Reilly Network; 07-08-2002; Disponível em http://www.oreillynet.com/pub/a/onjava/2002/08/07/j2eewebsvs.html; Acessado em 08-05-2003.

[10] NAGAPPAN, R, Skoczylas, R, Sriganesh, R. P.; "Developing Java Web Services : Architecting and Developing Secure Web Services Using Java"; John Wiley & Sons; Indianapolis – Indiana – USA; Fevereiro/2003.

[11] MCGOVERN, J.; Tyagi, S.; Stevens, M.; Mathew, S.; "Java Web Services Architecture"; Morgan Kaufmann; San Francisco – CA – USA, 2003.

[12] GRAHAM, Steve, et al.; "Building Web Services with Java : Making Sense of XML, SOAP, WSDL, and UDDI"; SAMS; Indina-USA, 2002.

[13] FEIBEL, Werner; "The Encyclopedia of Network"; Second Edition; Sybex; Alameda-USA, 1996.

[14] JEWELL, Tyler, CHAPPELL, Dave; "UDDI : Universal Description, Discovery, and Integration, Part 1"; Ther O'Reilly Network; Disponível em http://www.oreillynet.com/pub/a/onjava/excerpt/jws_6/index1.html; Acessado em 09-08-2003.

[15] NEWCOMER, Eric; "Describing Web Services : WSDL"; Addison Wesley Professional; 15-08-2003; Disponível em http://www.informit.com ; Home>Articles>Web Development>Web Services, Acessado em 02-09-2003.

[16] POTSS, Stephen, KOPACK, Mike; "Disadvantages and Pitfalls of Web Services"; Sams; 16-05-2003; Disponível em http://www.informit.com ; Home>Articles>Web Development>Web Services>XML, Acessado em 02-09-2003.

[17] MARINESCU, Floyd; "EJB™ Design Patterns"; Disponível em http://www.middleware-company.com/; Acessado em Novembro de 2002.

[18] MONDAY, Paul; "Hands-on Java Data Objects"; Disponível em http://ibm.com/developerWorks; Acessado em 11/09/2002.

[19] SNYDER, Bruce; "Get sarated with Castor JDO"; Disponível em http://ibm.com/developerWroks; Acessado em 29/08/2002.

[20] Sun MicroSystems; "Model-View-Controller Architecture"; Disponível em http://java.sun.com/blueprints/patterns/j2ee_patterns/model_view_controller/index.html; Acessado em 28/05/2002.

[21] DORSEY, Paul, et. al.; "Oracle JDeveloper: o manual oficial"; Campus, Rio de Janeiro, Brasil, 2001.

[22] BORBA, Paulo, ARAÚJO,Saulo, et. al.;"Progressive Implementation of Distribuited Java Aplication"; Disponível em http://www.di.ufpe.br/~phmb; Acessado em 06/05/2000.

[23] BORBA, Paulo, VIANA Euricelia;"Integrando Java com Bancos de Dados Relacionais"; Disponível em http://www.di.ufpe.br/~phmb; Acessado em 02/05/2000.

[24] GOULART JUNIOR, Fernando Silveira; "Arquitetura para banco de dados heterogêneos: soluções usando objetos distribuídos e Java"; Recife; Dissertação (mestrado) – Universidade Federal de Pernambuco. CCEN. Ciência da Computação, 1999.

[25] CASABIANCA, Michel; "More with Ant"; Oracle Magazine, p.70-73, Jan./Fev. 2003.

[26] AYER, Steve J.; "Object-Oriented Client/Server Application Development Using Objectpall And C++"; Mcgraw-Hill system design and implementation series, New York, 1995.

[27] CHIAVEGATTO, Myrza V.; "As Práticas do Gerenciamento da Informação - Estudo Exploratório na Prefeitura de Belo Horizonte"; Tese de Mestrado, FJP, 1999.

[28] COSTA, José M. Lomasso; "Sistemas de Suporte à Tomada de Decisão na Administração Pública : o caso da Prefeitura de Belo Horizonte"; Tese de Mestrado, FJP, 2000.

[29] EUGÊNIO, Marconi; "Inteligência Social em Administrações Municipais - um estudo de caso Prefeitura de Belo Horizonte"; Tese de Mestrado, UFMG, 1996.

[30] KOHAMA, Hélio; "Contabilidade pública - teoria e prática"; 5o. edição, Atlas, São Paulo, 1996.

[31] SIMON, Errol; "Distributed Information Systems : From Client/Server To Distributed Multimedia"; Mcgraw-Hill, Nova Iorque, 1996.

[32] Redmond Comunications Inc; "Directions On Microsoft – Plataforma de Desenvolvimento .NET"; edição impressa, 2002.

[33] AMBLER, Scott W.; "The Design of a Robust Persistence Layer for Relational Database"; Disponível em http://www.roni-intl.com; Acessado em 15/03/2003.

[34] FLEURY, Marc; "BLUE – Why I Love EJBs"; Disponível em http://www.jboss.com; Acessado em 23/01/2003.

[35] GOKUL, Seshadri; "J2EE Application Servers: What Makes a Good Product"; Disponível em http://www.informit.com; Acessado em 14/10/2002.

[36] GOKUL, Seshadri; "J2EE : What Is and Waht It's not"; Disponível em http://www.informit.com; Acessado em 27/05/2002.

[37] Sun MicroSystem; "Java 2 Platform, Enteprise Edition (J2EE)"; Disponível em http://java.sun.com/j2ee/overview.html; overview2.html; overview3.html Acessado em 14/10/2002.

[38] ALUR, Deepak, et. al; "Core J2EE patterns: as melhores práticas e estratégias de design"; Campus, Rio de Janeiro, Brasil, 2002.

[39] MORISSEAU-LEROY, Nirva, et. al.; "Oracle 8i : programação de componentes Java com EJB, CORBA e JSP"; Campus, Rio de Janeiro, Brasil, 2001.

[40] RICARTE, Ivan Luiz Marques; "Programação Orientada a Objetos : Uma Abordagem com Java"; Disponível em http://go.ricarte; Acessado em 07/09/2002.

[41] HUNTER, Jason; "How to get started with server-side Java"; Disponível em http://www.javaworld.com/javaworld/jw-03-1997/jw-03-ssj_p.html, Acessado em 02/02/2003.

[42] MONSON-HAEFEL, Richard; "Create forward-compatible beans in EJB, Part 1"; Disponível em http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-ejb1_p.html; Acessado em 14/11/2002.

[43] MONSON-HAEFEL, Richard; "Create forward-compatible beans in EJB, Part 2"; Disponível em http://www.javaworld.com/javaworld/jw-01-2000/jw-01-ssj-ejb2_p.html; Acessado em 14/11/2002.

[44] SESHADRI, Govind; "Understanding JavaServer Pages Model 2 architecture"; Disponível em http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmv2_p.html; Acessado em 14/11/2002.

[45] GOODWIL, James; "Installing and Configuring Tomcat"; Disponível em http://www.oreillynet.com/pub/a/onjava/2001/03/29/tomcat.html; Acessado em 05/12/2002.

[46] BATINI, Carlo, et. al. "Conceptual database design : an entry-relatioship approach"; The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, USA, 1992.

[47] ELMASRI, Ramez; "Fundamentals of database systems"; Addison-Wesley Publishing Company, Menlo Park, CA, USA, 1994.

[48] BRUCE, Thomas; "Designing quality databases : pratical infomation management & IDEF1X"; Dorset House Publishing, New York, NY, USA, 1992.

[49] PRESSMAN, Roger S.; "Software Engineering – A practitioner's Approach"; McGraw-Hill International Editions, Computer Science Series, Singapore, 1987.

[50] KURNIAWAN, Budi; "Java para a Web com Servlets, JSP e EJB"; Editora Ciência Moderna Ltda., Rio de Janeiro, 2002.

[51] ABOP; "Revista ABOP Edição Especial Glossário de Termos Orçamentários e Afins 33"; Associação Brasileira de Orçamento Público, Brasília, 3° quadrimestre de 1992.

[52] MACHADO JR., José Teixeira, et al.; "A Lei 4.320 comentada [por] J. Teixeira Machado JAX R. [e] Heraldo da Costa Reis", IBAM, Rio de Janeiro, 2000/2001.

[53] HORSTMANN, Cay; "CoreJava 2 Volume I – Fundamentos"; Campus, Rio de Janeiro, 2001.

[54] HORSTMANN, Cay; "CoreJava 2 Volume II – Fundamentos"; Campus, Rio de Janeiro, 2001.

[55] Sun Microsystem; "Using Web Services Effectively"; Disponível em http://java.sun.com/blueprints/webservices/using/webservbp.html; Acessado em 21/11/03;

[56] CRUZ, Flávio, et. al; "Comentários à Lei 4.320"; Editora Atlas, São Paulo, 1999.

[57] BODDOF, Stephanie, et. al.; "The J2EE Tutorial ver. 1.3"; Sun Microsystens, 2001, Disponível em http://java.sun.com; Acessado em 12/01/2002.

[58] SILVA, João Guilherme de Moraes; "Aplicações VoIP Utilizando o Teleporto da Rede Metropolitana da Prefeitura de Manaus"; Recife; Dissertação (mestrado) – Universidade Federal de Pernambuco. Pós-Graduação em Engenharia Elétrica do Centro de Tecnologia e Geociências, 2003.

[59] MAZZETI, Gerardo, MINK, Carlos; "HTML 4 com XML"; Makron Books, São Paulo, 2000.

[60] Oracle; "Buffer Overflow in the XML Database of Oracle 9i Database Server"; Oracle Security Alert 58; Dated : 18 August 2003; Severity 1; Disponível em http://www.oracle.com; Acessado em : 20/12/2003.

[61] SHERMAN, Roby; "Oracle 9i Rel 2 – XDB Port Nightmares" ; Disponível em http://www.interealm.com/technotes/roby/xdb_ports.html; Acessado em : 22/12/2003.

[62] ENAP; "Formação de Multiplicadores do Novo Modelo de Planejamento, Gestão e Orçamento"; Brasília, Distrito Federal; 2003.

[63] ARMSTRONG, Eric, et. al; "The Java Web Services Tutorial", October 9, 2003; Disponível em http://java.sun.com/docs/books/tutotial/javawstutorial.pdf; Acessado em 26/11/2003.

[64] IBM, Microsoft; "Security in a Web Services World : A Proposed Architecture and Roadmap"; April, 7, 2002, Version 1.0; Disponível em http://www-106.ibm.com/developeworks/library/Web Services-secmap; Acessado em : 11/4/2002.