UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Departamento de Engenharia de Estruturas
Curso de Pós-Graduação em Engenharia de Estruturas

UM SERVIÇO WEB PARA O MÉTODO DOS ELEMENTOS FINITOS

Luciana Sampaio Camara

Dissertação apresentada ao curso de Pós-Graduação em Engenharia de Estruturas da UNIVERSIDADE FEDERAL DE MINAS GERAIS, como parte dos requisitos para obtenção do título de MESTRE EM ENGENHARIA DE ESTRUTURAS.

Orientador: Prof. Roque Luiz da Silva Pitangueira

Belo Horizonte
Agosto de 2007

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ESTRUTURAS

"UM SERVIÇO WEB PARA O MÉTODO DOS ELEMENTOS FINITOS"

Luciana Sampaio Câmara

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Estruturas da Escola de Engenharia da Universidade Federal de Minas Gerais, como parte dos requisitos necessários à obtenção do título de "Mestre em Engenharia de Estruturas".

Comissão Examinadora:

_____
Prof. Dr. Roque Luiz da Silva Pitangueira
DEES - UFMG - (Orientador)

_____
Prof. Dr. Felicio Bruzzi Barros
DEES - UFMG

_____
Prof. Dr. Rodrigo Weber dos Santos
UFJF

_____
Prof. Dr. Flávio de Souza Barbosa
UFJF

Belo Horizonte, 31 de agosto de 2007

Não há razão no mundo para alguém querer um computador em sua casa. Nenhuma razão.

*(*Ken Olsen, Presidente da Digital Equipment Corp, 1977*)*

O homem está sempre disposto a negar tudo aquilo que não compreende.

*(*Blaise Pascal*)*

O único lugar onde o sucesso vem antes do trabalho é no dicionário.

*(*Albert Einstein*)*

Dedico este trabalho a meus queridos pais.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Códigos

Resumo

A evolução e popularização da Internet levam a utilização da mesma como uma plataforma, um ambiente que oferece programas e onde pode-se armazenar e acessar arquivos. Os Serviços Web surgem, neste cenário, como uma solução à crescente necessidade de se trocar informações entre sistemas diferentes através de padrões simples e públicos estabelecidos na Internet (*XML*, *HTTP*, *TCP/IP*) e em tecnologias abertas como *WSDL*, *UDDI* e *SOAP*.

O **INSANE** é um sistema computacional de análise de modelos discretos de elementos finitos. Esta dissertação de mestrado teve como objetivo disponibilizar o seu núcleo numérico como um *Serviço Web* para resolução de modelos de elementos finitos através da Internet.

O Serviço Web desenvolvido pode ser consumido por aplicações diversas (aplicações Web, interfaces gráficas de pré e pós-processamento para "*Desktop*", aplicações para dispositivos microeletrônicos como palms e celulares, dentre outros), pois agrega ao sistema as características de baixo acoplamento, interoperabilidade, reutilização e flexibilidade da tecnologia de Serviços Web.

A atualização do sistema acontece de maneira transparente para o usuário, pois a implementação do sistema está concentrada em um único lugar, o servidor onde o serviço está hospedado. A utilização deste serviço pode evitar a constante re-implementação de recursos já consolidados e disponíveis em programas de elementos finitos.

São apresentados três exemplos de consumidores do Serviço Web **INSANE**, sendo que um deles, o cliente Web, permite o uso do serviço pela Web, sem a necessidade de nenhum programa além do navegador.

Abstract

The evolution and popularization of the Internet lead to its utilization as a platform, an environment that offers softwares and where it's possible to save and access files. The Web Services appear, at this scenario, as a solution to the increasing need of exchanging informations among different systems through simple and public Internet's established standards (*XML*, *HTTP*, *TCP/IP*) and open technologies such as *WSDL*, *UDDI* and *SOAP*.

The **INSANE** is a computational system of finite element models analysis. The goal of this master's thesis was to make its numerical kernel available as a Web Service to solve finite element models through the Internet.

The developed Web Service may be consumed by different applications (Web applications, Desktop pre and post-processing graphics interfaces, microelectronics devices such as PDA and cellular phones, among others), since it adds to the system the Web Services features such as loose coupling, interoperability, reuse and flexibility.

The system's updates occur in a transparent way to the user, since its implementation is concentrated at a single place, the server where the service is hosted. This service's use can avoid the frequent re-implementation of already consolidated resources, available in finite element method softwares.

Three examples of **INSANE** Web Service's consumers are presented and one of them, the Web client, allows its use through the Internet, without the need of any other software besides the Web browser.

Agradecimentos

A *DEUS* pelo amparo espiritual em todos os momentos de minha vida.

A *meus pais* pela formação de meu caráter, pela minha vida e por todos os ensinamentos.

Ao professor, orientador e amigo *Roque Luiz da Silva Pitangueira*, agradeço pelo tema de meu trabalho de mestrado, por acreditar em minha capacidade e, especialmente, por todo o apoio e incentivo que recebi durante estes anos de estudo e que permitiram essa conquista. Agradeço, também, pela vibração em cada dificuldade superada neste trabalho.

Ao professor, chefe e amigo *Estêvão Bicalho Pinto Rodrigues*, agradeço pelo apoio, incentivo e exemplos, como de um pai, fundamentais nesta caminhada. Agradeço também pela compreensão ao meu horário de trabalho reduzido que a dedicação ao mestrado exigiu.

A todos os *Insanos* pelo companheirismo e pela amizade que surgiu ao longo destes anos. Em especial, agradeço à Jamile pelas lições de LaTeX, ao Samuel pela ajuda na parte gráfica do **INSANE** e ao Flavio pela disponibilidade na configuração do laboratório e ajuda com a impressão desta dissertação.

Aos irmãos *Krishna, Maíra e Mael Caldas* pela orientação, ajuda, conselhos e ensinamentos passados. Ao colega virtual *Thilina Gunarathne*, do Sri Lanka, pela ajuda em uma hora crucial da implementação do Serviço Web. Benefícios da vida moderna (Internet).

Aos *professores* e *funcionários* do Departamento de Engenharia de Estruturas.

À *FAPEMIG* pelo apoio financeiro na forma de projeto de pesquisa, que permitiu a montagem do Laboratório de Software Livre do grupo **INSANE**.

Aos meus *amigos e todos aqueles* que de alguma forma contribuíram para a realização deste trabalho.

Capítulo 1

INTRODUÇÃO

Nos últimos anos, com a popularização dos computadores e o aumento de recursos e tecnologias, a engenharia tem tido à sua disposição várias ferramentas computacionais de análise, dimensionamento e detalhamento de estruturas. Programas de cálculo cada vez mais poderosos e rápidos permitem a modelagem e análise de estruturas complexas oferecendo modos amigáveis de entrada de dados e visualização de resultados. Utilizando-se o Método dos Elementos Finitos (MEF), estes programas permitem analisar estruturas de geometrias diversas, constituídas por diferentes materiais e sujeitas a diferentes solicitações.

A facilidade de acesso à informação proporcionada pela Internet incentiva cada vez mais o desenvolvimento de serviços e aplicações "*online*". Vários "*sites*" vêm surgindo na rede, disponibilizando aplicativos que não precisam ser instalados no computador pessoal e nem comprados para serem utilizados. Editores de texto, leitores de "*e-mail*[1]", editores de imagem, planilhas eletrônicas e até mesmo agenda pessoal já podem ser encontrados e acessados gratuitamente. Um exemplo desses serviços é o portal **Google** (Google, 2007) que oferece agenda (Figura 1.1) e planilha eletrônica (Figura 1.2) "*online*" e de uso gratuito.

O acesso à Internet, que há alguns anos era muito lento e caro, está se tornando mais acessível à medida em que vários provedores oferecem acesso em banda larga (tornando-o extremamente mais rápido) a preços fixos, e a concorrência entre eles ocasiona uma forte queda de preços.

[1]Correio eletrônico.

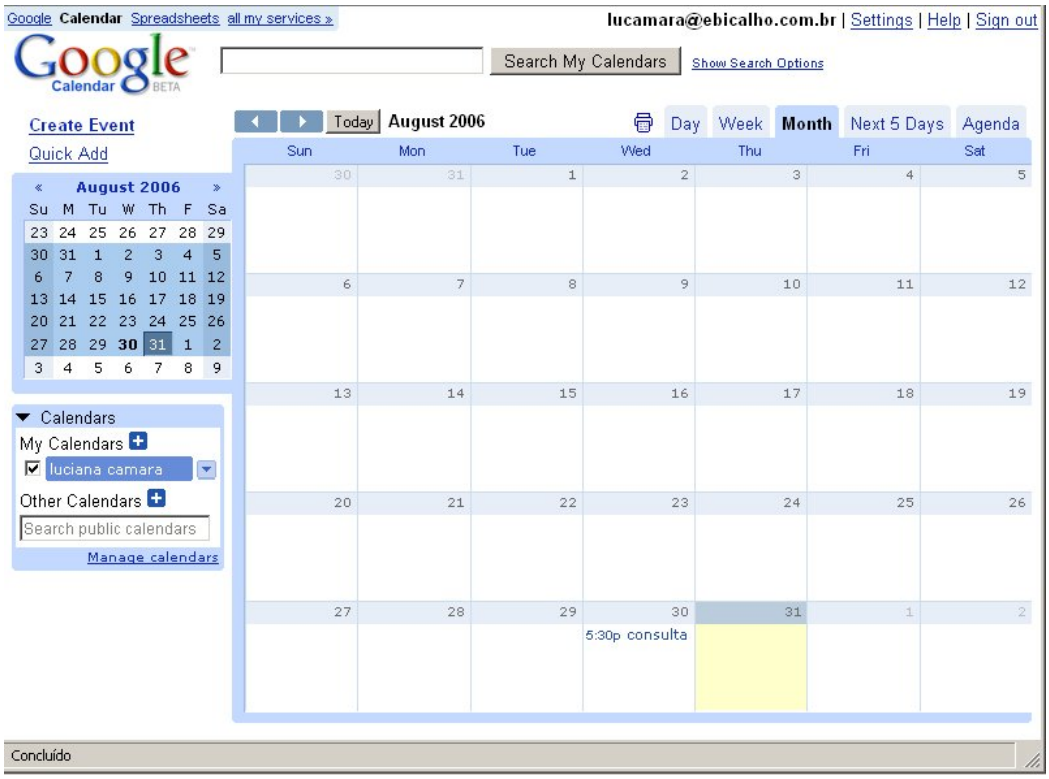
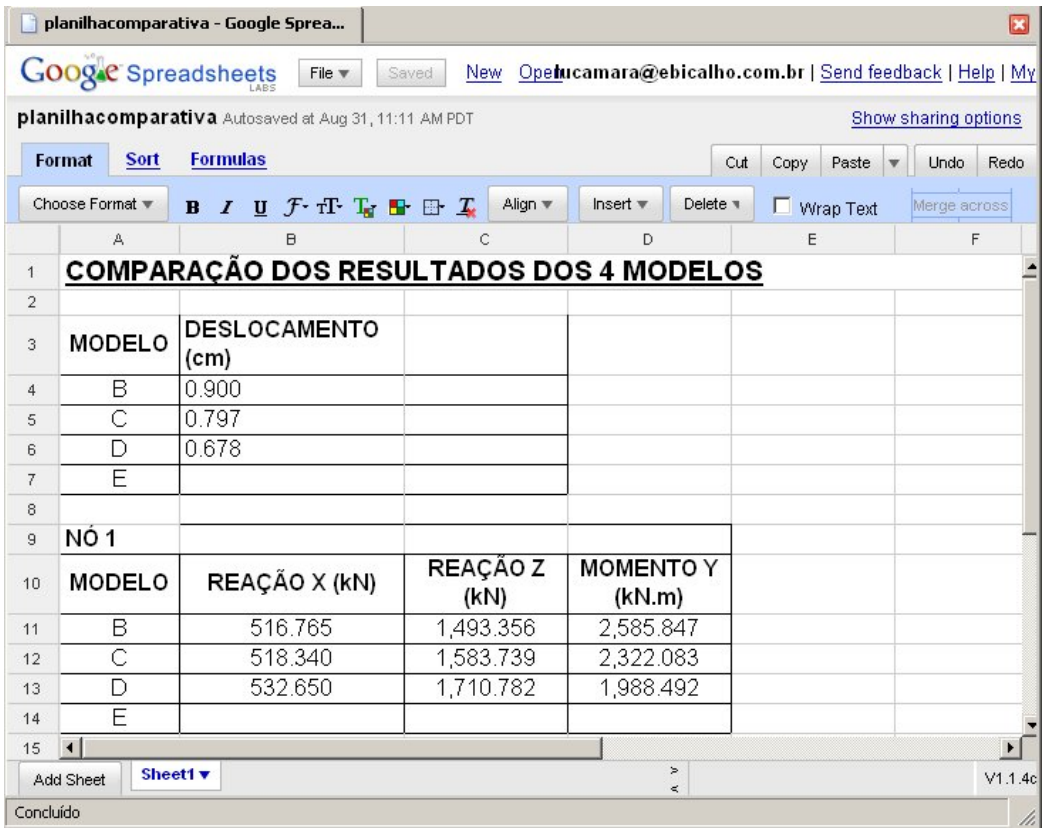


Figura 1.1: Agenda "*Online*" do portal Google (Google, 2007).



	A	B	C	D	E	F
1	**COMPARAÇÃO DOS RESULTADOS DOS 4 MODELOS**					
2						
3	MODELO	DESLOCAMENTO (cm)				
4	B	0.900				
5	C	0.797				
6	D	0.678				
7	E					
8						
9	NÓ 1					
10	MODELO	REAÇÃO X (kN)	REAÇÃO Z (kN)	MOMENTO Y (kN.m)		
11	B	516.765	1,493.356	2,585.847		
12	C	518.340	1,583.739	2,322.083		
13	D	532.650	1,710.782	1,988.492		
14	E					
15						

Figura 1.2: Planilha eletrônica "*Online*" do portal Google (Google, 2007).

A Internet cada vez mais deixa de ser vista e usada como uma simples rede de computadores para ser uma plataforma, um ambiente que oferece aplicações e onde pode-se armazenar arquivos e, conseqüentemente, acessá-los de qualquer lugar que ofereça acesso à grande rede.

Poder usar a Internet para obter acesso à versão mais atualizada de um programa (à medida em que novos recursos forem acrescentados a ele) sem a necessidade de reinstalá-lo, parece ser muito interessante, principalmente para sistemas computacionais em constante desenvolvimento.

1.1 Objetivos do Trabalho

1.1.1 Objetivos Gerais

O projeto **INSANE**, em realização no Departamento de Engenharia de Estruturas da Universidade Federal de Minas Gerais (`http://insane.dees.ufmg.br`), visa desenvolver um sistema computacional de análise de modelos discretos de elementos finitos. Utilizando-se modernos recursos tecnológicos, o sistema, a cada novo trabalho de um colaborador, amplia sua complexidade através da incorporação de uma nova aplicação, modelo, interface ou solução desenvolvida no seu trabalho. É importante ressaltar, que a cada evolução o sistema é expandido sem a necessidade de se refazer nenhuma implementação previamente desenvolvida.

O sistema **INSANE** é formado por aplicativos que podem ser classificados em três grandes segmentos: pré-processadores, processador e pós-processadores implementados em JAVA.

O presente trabalho ampliou a complexidade do sistema através do desenvolvimento de aplicativos que atuam nos segmentos de processadores e pós-processadores do sistema **INSANE**.

1.1.2 Objetivos Específicos

O objetivo específico dessa dissertação foi desenvolver um *Serviço Web* para modelos do método dos elementos finitos. O Serviço em questão usa as classes Java dos projetos **INSANE** do núcleo numérico, para resolver estes modelos a partir de um arquivo de dados e retorna os

arquivos de resultados gerados.

Este Serviço Web possui características que permitem a sua utilização por aplicativos desenvolvidos em qualquer linguagem e executados em qualquer plataforma e em qualquer lugar através da Internet.

Para a pesquisa na área de métodos numéricos e computacionais, a utilização deste serviço pode evitar a constante re-implementação de recursos já consolidados e disponíveis em programas de elementos finitos.

1.2 Organização do Texto

Este trabalho está organizado em 8 capítulos, além do Capítulo 1 que dá uma visão geral sobre o trabalho desenvolvido.

No Capítulo 2 são apresentados os principais conceitos relacionados à computação distribuída e, em seguida, são expostos alguns de seus modelos tradicionais que antecederam o modelo de Serviços Web.

O Capítulo 3 expõe a nova tecnologia de Serviços Web, apresentando o seu conceito e as vantagens oferecidas por ela que a leva a ser considerada uma promissora opção na área de computação distribuída. Além disso, é discutido o processo de desenvolvimento de um Serviço Web e, ao final, são apresentados alguns trabalhos científicos relacionados a esta tecnologia e à engenharia civil.

Em seguida, são apresentadas no Capítulo 4 as principais tecnologias necessárias ao desenvolvimento do Serviço Web proposto neste trabalho de dissertação e, no Capítulo 5, as tecnologias relacionadas a Aplicações Web que são utilizadas em um dos consumidores do Serviço Web.

O Capítulo 6 apresenta especificamente o processo de desenvolvimento do Serviço Web **INSANE**(***InsaneService***). Inicialmente são discutidos a generalização do modelo do sistema e o projeto orientado a objetos do seu núcleo numérico. São detalhadas as fases de codificação, disponibilização e operação do Serviço desenvolvido, assim como as características de futuras atualizações do mesmo.

No Capítulo 7 são apresentados três diferentes consumidores (clientes) com o intuito de demonstrar o uso do Serviço Web desenvolvido. Foram implementados clientes Java padrão, *Swing* e, também uma aplicação Web, que já está disponível para uso. São utilizados exemplos para a demonstração dos recursos dos clientes e para ilustrar o funcionamento do consumo do Serviço Web **INSANE**.

Conclusões, principais contribuições oferecidas por este trabalho, bem como algumas sugestões para futuros trabalhos de pesquisa envolvendo o tema discutido, são apresentadas no Capítulo 8.

Devido ao constante uso de termos técnicos pertencentes à área de informática, bem como siglas e palavras em inglês, ao final da dissertação é apresentado um glossário. As palavras presentes neste glossário foram escritas em fonte *mono espaçada e em estilo itálico* para diferenciá-las.

Capítulo 2

MODELOS DE COMPUTAÇÃO DISTRIBUÍDA

2.1 Introdução

O fortalecimento e o uso disseminado das redes de computadores (pessoais e de grande porte) como recurso computacional distribuído fez com que a computação em rede ganhasse importância e disponibilizasse as chamadas remotas a procedimentos (*RPC* ou Remote Procedure Calls) sobre o protocolo de rede denominado *TCP/IP* (Transmission Control Protocol/Internet Protocol), as quais foram amplamente aceitas como modo de comunicação entre aplicações (Albinader e Lins, 2006). Assim, computação distribuída pode ser entendida como um sistema no qual seus diferentes componentes podem estar localizados geograficamente em diferentes computadores conectados em rede. A Figura 2.1 representa, de modo simplificado, um modelo de computação distribuída que tem como base a rede mundial de computadores (Internet). A computação distribuída possibilita à aplicação o acesso a objetos funcionais localizados em qualquer lugar da rede.

Um ambiente de computação distribuída traz muitas vantagens em relação ao ambiente tradicional, no qual um computador opera sozinho executando programas sem conexões externas, dentre as quais, pode-se citar:

1. Alto desempenho: aplicações podem ser executadas em paralelo e distribuir a carga de processamento entre vários computadores;

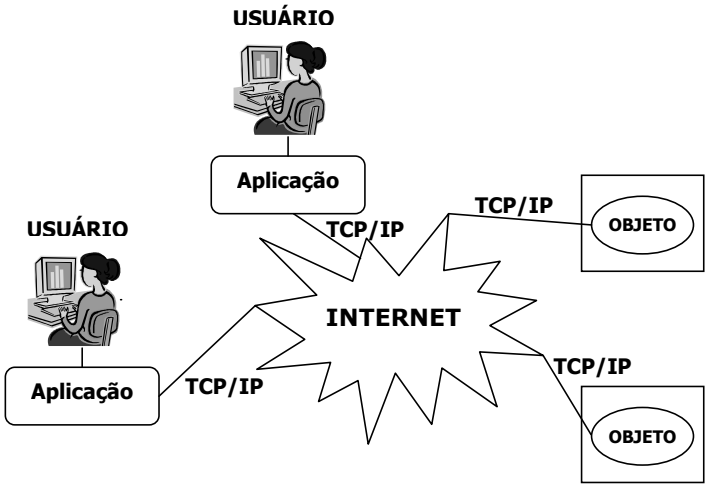


Figura 2.1: Modelo de computação distribuída na Internet.

2. Colaboração: muitas aplicações podem ser conectadas a mecanismos de computação distribuída;

3. Alta confiabilidade e disponibilidade: aplicações e servidores podem ser implantados em conjuntos redundantes (se um vier a falhar haverá outro capaz de responder à solicitação feita ao que falhou);

4. Escalabilidade e extensibilidade: novos componentes podem ser adicionados ao sistema e o mesmo pode ser reconfigurado de acordo como os recursos disponíveis na rede;

5. Alta produtividade: a aplicação pode ser dividida em partes que podem ser desenvolvidas por equipes diferentes de maneira isolada;

6. Reutilização: componentes distribuídos podem ser utilizados por vários clientes em diversas aplicações;

7. Custo reduzido de desenvolvimento devido ao alto grau de reutilização de seus componentes.

Vários modelos de computação distribuída foram e são utilizados até hoje. Nas próximas seções serão expostos alguns destes modelos, com o propósito de situar o surgimento dos Serviços Web como opção de computação distribuída (este tópico será discutido mais detalhadamente no Capítulo 3).

2.2 Cliente-Servidor

Os anos iniciais das aplicações de negócios distribuídas foram dominados pelo modelo de duas camadas, conhecido como Cliente-Servidor.

A Figura 2.2 ilustra este modelo, no qual a camada mais próxima do usuário, denominada cliente, possui uma aplicação responsável pela interface com o usuário e também pela lógica de negócio. A segunda camada, denominada servidor, tinha a função de gerenciar a organização e o armazenamento de dados da aplicação. Funcionava em um computador central da rede, geralmente mais robusto.

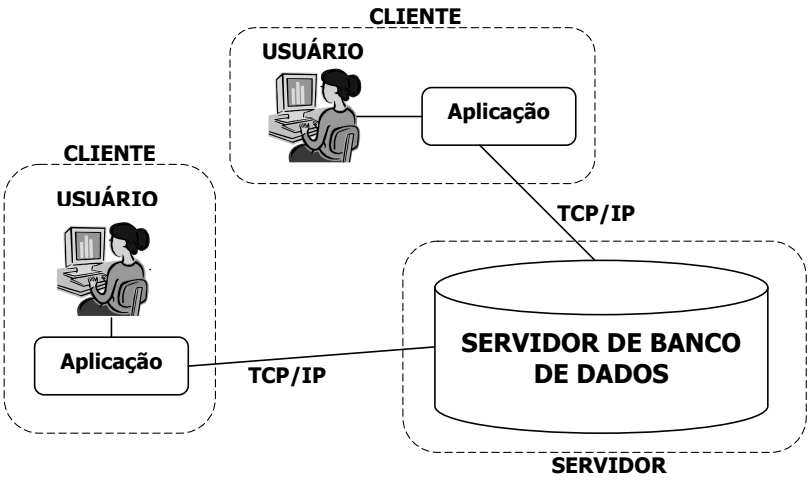


Figura 2.2: Modelo Cliente-Servidor.

Este modelo se caracteriza pelo forte acoplamento entre as partes. A manutenção e atualização dos clientes torna-se trabalhosa, pois cada cliente deve ser atualizado e mantido separadamente. Todavia, ele se tornou bastante popular e foi largamente adotado em sistemas corporativos (Albinader e Lins, 2006).

2.3 CORBA

O padrão *CORBA* (*Common Object Request Broker Architecture*) constitui-se uma tentiva 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.

Este modelo segue o padrão de orientação a objetos e apresenta independência em relação a protocolos, sistemas operacionais, linguagens de programação e plataformas de hardware. Entretanto, é preciso que a aplicação mapeie sua interface para o IDL (*Interface Definition Language*), que é uma linguagem neutra, desenhada para a disponibilização e acesso a serviços (métodos e funções) de objetos remotos *CORBA* (Albinader e Lins, 2006).

Este padrão ainda define uma coleção de serviços em nível de sistema para a manipulação de aplicações de baixo nível como ciclo de vida, persistência, transação, nomeação e segurança.

O comportamento de uma solução que adota o *CORBA* é baseado em um ORB (*Object Request Broker*), que é um objeto funcionando como um mecanismo transparente, que recebe solicitações e manda respostas aos objetos da rede, independente de qual ambiente estes objetos estejam (adicionando interoperabilidade ao padrão).

A Figura 2.3 mostra uma representação simplificada da arquitetura *CORBA*, funcionando como catalisadora de recursos entre aplicações escritas em linguagens diferentes (C, C++ e Java), interoperando através da IDL. A IDL é empregada para estabelecer contratos, limitando a interação das aplicações e estabelecendo as interfaces com os clientes. O ORB funciona como um objeto corredor, uma ponte, disponibilizando a infra-estrutura de comunicação para enviar e receber solicitações e respostas dos clientes para os servidores.

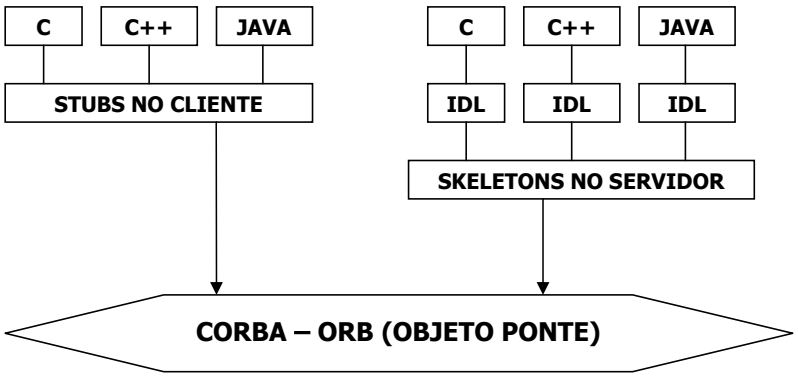


Figura 2.3: Modelo *CORBA*.

Apesar de trazer interoperabilidade, reuso de aplicações (estas podem ser disponibilizadas como objetos que os clientes podem invocar através do ORB) entre outras facilidades à computação distribuída, o padrão *CORBA* teve baixa disseminação e possui poucos produtos que o

implementam. Isto se deve basicamente a três fatores (Albinader e Lins, 2006):

(i) Investimento inicial elevado para implantação e treino da equipe de desenvolvimento;

(ii) Baixa disponibilidade de implementações estáveis *CORBA*: tradicionalmente, implementações *CORBA* tem sido difíceis de se obter, além de serem complexas, incompletas e caras;

(iii) Baixa escalabilidade: devido a alta e rigorosa natureza do acoplamento orientado a conexão, volumes de acesso muito elevados podem fazer com que as respostas se tornem lentas.

2.4 Java RMI

A "*Sun MicroSystems*" desenvolveu o *RMI* (*Remote Method Invocation*), que permite aplicações Java chamarem remotamente objetos, passarem argumentos a eles e receberem valores de retorno. O mecanismo de serialização de objetos Java é empregado para converter objetos em cadeias de bits que podem, então, ser transportados pela rede e remontados no seu destino (Albinader e Lins, 2006).

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. Isto caracteriza um cenário de computação distribuída, já que "VMs" podem funcionar em diferentes computadores da rede.

A Figura 2.4 apresenta de modo simplificado o modelo de arquitetura de uma aplicação baseada em Java *RMI*.

Aplicações distribuídas utilizando *RMI* possuem três limitações consideradas importantes:

(i) São limitadas à plataforma Java;

(ii) São fortemente acopladas, devido à sua natureza orientada à conexão, o que dificulta a sua escalabilidade;

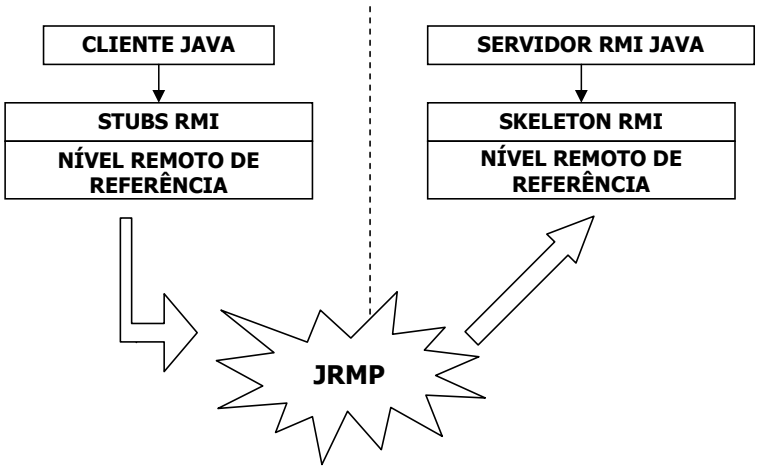


Figura 2.4: Modelo Java *RMI*.

(iii) Não possuem mecanismos específicos para gerenciamento de sessão, o que faz com que a aplicação seja limitada a alguns domínios de aplicação.

2.5 Microsoft DCOM

A "*Microsoft*" definiu o COM (*Component Object Model*), que utiliza outro mecanismo proprietário denominado OLE (*Object Linking and Embedding*) para a comunicação entre aplicações *Windows* baseadas em componentes, por meio de binário e em redes do padrão do sistema operacional *Windows*. Esta solução, entretanto, demonstrou grande instabilidade e baixa confiabilidade (Albinader e Lins, 2006).

A tecnologia da Microsoft para a computação distribuída na plataforma *Windows* denomina-se *DCOM* (*Distribuited Common Object Model*), que implementa o mecanismo *RPC* através do qual as aplicações COM podem comunicar-se empregando o protocolo *DCOM*.

O *DCOM* expõe através de interface definida (através dos *stub*s e *skeleton*s, que são interfaces que encapsulam os procedimentos de conexão remota) os métodos de objetos COM que podem ser invocados remotamente através da rede. Assim, aplicações clientes podem invocar métodos de objetos COM (em computadores remotos) como se estes estivessem presentes localmente.

A Figura 2.5 demonstra simplificadamente o modelo de arquitetura *DCOM*.

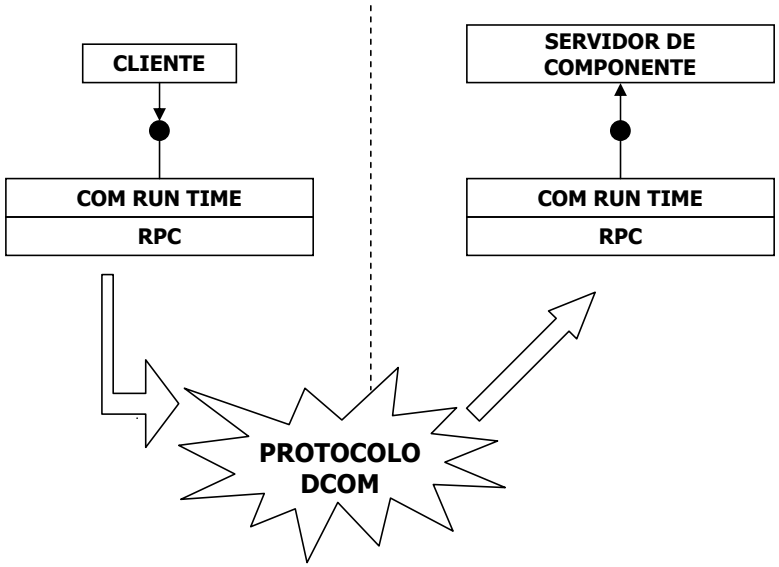


Figura 2.5: Modelo Microsoft *DCOM*.

O *DCOM* tem obtido aprovação e aceitação para sistemas baseados na plataforma *Windows*, porém, apresenta também limitações, entre as quais pode-se citar:

(i) Apresenta plataforma proprietária e pertencente a um único fornecedor (*Windows* da *Microsoft*);

(ii) Não há gerenciamento de estado da informação distribuída;

(iii) Baixa escalabilidade e alta complexidade no gerenciamento de sessões.

2.6 Modelo Message-Oriented Middleware

O modelo *MOM* (*Message-Oriented Middleware*) é baseado em baixo acoplamento e comunicação assíncrona, no qual o cliente não necessita saber nada sobre os recipientes da aplicação servidora ou quais métodos devem ser chamados. Este modelo 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 de 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 aguardar a resposta da aplicação destino.

Na arquitetura *MOM*, uma aplicação interage com a infra-estrutura de mensagens através de adaptadores personalizados para cada ambiente de aplicação. Existem várias soluções proprietárias para este modelo de computação distribuída. Uma das mais interessantes é o JMS (*Java Message Service*), que possibilita comunicação ponto a ponto, sistema de mensagens baseado em publicação/inscrição, capacidade de gerenciar transações, confiabilidade na entrega de mensagens e segurança.

A Figura 2.6 apresenta simplificadamente o modelo de arquitetura *MOM*.

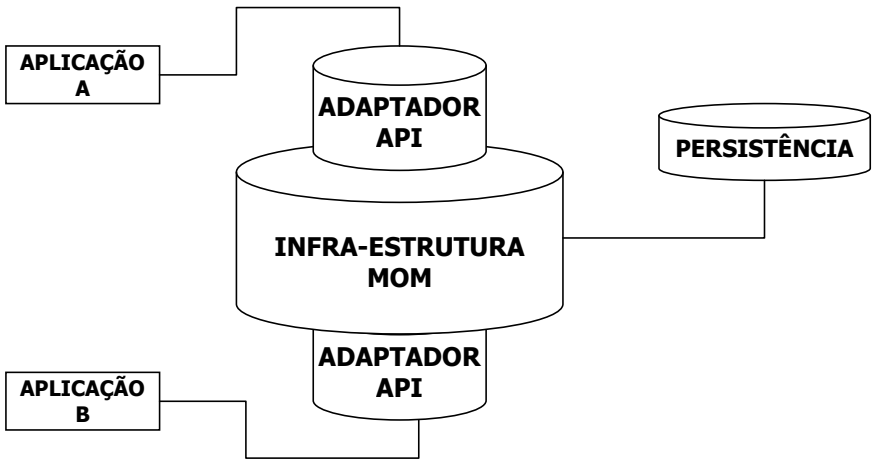


Figura 2.6: Modelo *MOM* (Message-Oriented Middleware).

Sistemas de aplicações distribuídas baseados em *MOM* apresentam alguns desafios, entre os quais pode-se citar:

(i) A maioria dos padrões de *MOM* implementados possuem *API*s nativas (ver Figura 2.6) usadas para a comunicação com o núcleo de sua infra-estrutura, o que afeta a portabilidade das aplicações entre diferentes implementações e pode prendê-la a um fornecedor específico;

(ii) As mensagens utilizadas para integrar aplicações neste modelo 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.

2.7 Modelo J2EE

A evolução da Internet e seus padrões impuseram um modelo de programação no qual o cliente deve ser o navegador Web e o servidor deve resolver toda a complexidade da aplicação, além de responder a solicitação no mais simples *HTML* (seção 5.2) possível. Para atender a esta demanda, a *J2EE* (*Java 2 Enterprise Edition*) disponibiliza um modelo de programação baseado na Web e em componentes de negócios gerenciados por um servidor de aplicação *J2EE*. Esse servidor de aplicação constitui-se em um conjunto de *API*s e serviços de nível básico (segurança, suporte a transação, gerenciamento de conexões, serviços de concorrência, entre outros) disponíveis para os componentes.

Este modelo é baseado em Java e em padrões e especificações da indústria, disponibilizando interfaces para a conexão com vários sistemas legados e sistemas de informação. Possui capacidade de conexão com vários tipos de clientes, como PDA's (*Personal Digital Assistants*) ou navegadores Web. Além disso, permite conexões com clientes considerados "Ricos", como *applets* Java, *CORBA* e aplicações Java sendo executados em computadores pessoais ligados em rede através do protocolo IIOP (*Internet Inter−ORB Protocol*).

O Servidor *J2EE* possui uma arquitetura típica dividida em três camadas lógicas, como ilustra a Figura 2.7, as quais separam claramente os vários componentes da aplicação.

A camada de apresentação aloja a parte de interação com o cliente. Nesta camada estão as páginas *HTML*, JSP (*Java Server Pages*), os *Servlets*, as imagens e demais artefatos que devem ser transferidos aos clientes. Esta camada é responsável por emitir respostas e extrair da camada de aplicação as informações necessárias na geração de respostas às solicitações do cliente.

A camada de aplicação é responsável pela lógica, pelas restrições, regras de negócio e modelagem de componentes das aplicações em *J2EE* e é onde são hospedados os EJBS (*Enterprise Java Beans*), classes Java especiais que têm a função de extrair dados da camada de integração.

A camada de integração compreende as fontes de dados que podem ser arquivos comuns, banco de dados, aplicações antigas, entre outras. Esta camada tem a função de fornecer dados para a aplicação e também realizar a persistência dos mesmos.

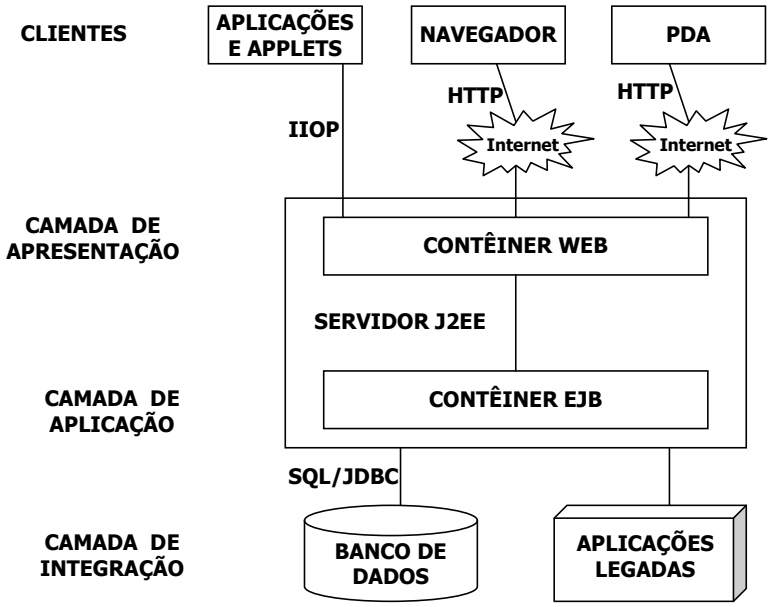


Figura 2.7: Modelo *J2EE*.

O emprego de *XML* (*Extensible Markup Language*), discutido na seção 4.3, como meio de troca de dados estruturados entre aplicações, contribui para aumentar a interoperabilidade entre elas e acrescenta escalabilidade às aplicações. A combinação de *XML* e do padrão *J2EE* oferece uma alternativa completa para a interoperabilidade entre aplicações de negócios.

Capítulo 3

SERVIÇOS WEB

3.1 Introdução

Os Serviços Web surgiram como conseqüência natural da evolução da Internet. A popularização deste meio de comunicação leva à evolução das tecnologias envolvidas, o que possibilita o aparecimento de aplicações disponibilizadas ao público em grande escala.

Inicialmente, a Internet era constituída por páginas estáticas que faziam pouco mais que exibir informações através de programas denominados navegadores. Atualmente, as páginas disponibilizadas são capazes de interagir, acionar programas geradores de informações dinâmicas, provenientes de banco de dados e outras fontes. Os usuários, antes meros espectadores, agora interagem, inserem, alteram e excluem informações. O navegador Web tornou-se o cliente universal da Internet.

Neste cenário, cresce a necessidade de se trocar informações entre sistemas diferentes e que eles possam se comunicar através de padrões simples e públicos.

3.2 Conceito

Segundo a W3C (*World Wide Web Consortium*), um Serviço Web é (Booth et al., 2004):

> "*um sistema de software planejado para suportar a interação máquina a máquina em uma rede. Ele tem uma interface descrita em um formato processável de máquina (especificamente* `WSDL`*). Outros sistemas interagem com um Serviço Web*

de maneira prescrita por seu descritor usando mensagens SOAP, tipicamente transportadas usando HTTP e uma serialização XML em conjunto com outras tecnologias relacionadas à Web".

Ou seja, é uma tecnologia idealizada para comunicação entre sistemas. Um Serviço Web é uma noção abstrata que deve ser implementada por um agente (cliente). Este agente pode ser uma parte de "*software*" ou "*hardware*" que envia e recebe mensagens. Como o serviço é bem definido pelo seu descritor (arquivo *WSDL* ou "Web Service Description Language"), ele pode ser implementado por diversos clientes em diferentes linguagens de programação e em diferentes plataformas.

3.3 Serviços Web como opção para computação distribuída no ambiente da Internet

Conforme discutido no Capítulo 2, as arquiteturas tradicionais de sistemas distribuídos apresentam relativa fragilidade, pois acoplam fortemente vários componentes ao sistema ou são soluções proprietárias e complexas. Assim, demonstram-se altamente sensíveis a mudanças e apresentam dificuldade em acompanhar a dinâmica da Internet.

Os Serviços Web surgem como uma solução a esta demanda. Baseados em padrões estabelecidos na Internet (*XML*, *HTTP*, *TCP/IP*) e em tecnologias abertas como *WSDL*, *UDDI* e *SOAP* (discutidos no Capítulo 4), os Serviços Web constituem-se em *software* de baixo acoplamento, reutilizáveis, com componentes feitos para serem facilmente acessados pela Internet, representando, assim, um modo para integrar tarefas que compõem um processo de negócio através da Internet no qual procedimentos estão interligados para atingir um resultado concreto final (Albinader e Lins, 2006).

Entre as vantagens obtidas com o uso dos Serviços Web, pode-se citar (Nascimento, 2005) e (Albinader e Lins, 2006):

1. Interoperabilidade: é a característica mais importante desta tecnologia. Os Serviços Web podem ser implementados por diversos clientes em diferentes linguagens de programação

e em diferentes plataformas, ou seja, oferecem independência de plataforma de *hardware* e *software* (baixo acoplamento);

2. Reuso: aplicações disponibilizadas como Serviços Web podem ser facilmente utilizadas em outras aplicações;

3. Flexibilidade: o encapsulamento de implementações como Serviços Web permite que elas possam ser modificadas ou substituídas com simplicidade;

4. Interação simples entre clientes e servidores, pois os Serviços Web são baseados em padrões simples, públicos e amplamente testados, como o *HTTP* e *HTML*;

5. Um sistema baseado em Serviços Web funciona de modo descentralizado, sem a necessidade de uma coordenação central;

6. O destino das mensagens usadas em Serviços Web é especificado indiretamente com o uso de uma *URL* (*Universal Resource Locator*), propiciando a implementação de balanceamento de carga e controle de sessão;

7. Existência de várias implementações de seus protocolos disponíveis no mercado: já é possível utilizar *API*s de Serviços Web para várias linguagens (Java, C++, VBScript, *JavaScript*, .Net, Perl, PHP (Systinet, 2005)) sendo que alguns são de uso gratuito, como o *Axis* da linguagem Java (ver seção 4.7). Uma lista destas ferramentas pode ser obtida no *site* `http://en.wikipedia.org/wiki/List_of_Web_service_Frameworks`.

A tecnologia de Serviços Web apresenta, também, algumas limitações inerentes ao seu ambiente de operação, a Internet, entre as quais pode-se citar (Albinader e Lins, 2006) e (Sumra e Arulazi, 2007):

1. Descoberta e Localização: se o local onde o Serviço Web estiver hospedado mudar, os clientes precisarão atualizar o endereço de acesso ao mesmo. O registro do Serviço Web em um repositório de registros *UDDI* (ver seção 4.6) e a sua atualização podem ajudar os clientes na descoberta do novo endereço;

2. Confiabilidade no ambiente de Internet: devido ao fato dos serviços Web utilizarem protocolos como o *HTTP* (ver seção 4.2), não há garantia de que as mensagens serão entregues ao destino;

3. Segurança: serviços Web disponíveis para uso público podem não possuir nenhum esquema de segurança. A adoção de protocolos seguros (HTTPS, SSL, entre outros), criptografia, autenticação deve ser pensada em aplicações que necessitem de níveis maiores de segurança;

4. Transações: transações de longa duração (que podem consumir horas ou até mesmo dias) devem ser gerenciadas de alguma forma nos serviços Web;

5. Escalabilidade: mecanismos como balanceamento de carga devem ser empregados para o aumento da capacidade do serviço, em termos de quantidade de atendimentos e velocidade de resposta;

6. Desempenho: a vazão (número de solicitações respondidas em um determinado período de tempo) e a latência (tempo gasto entre o envio de uma solicitação e o recebimento de sua resposta) dependem da lógica da aplicação, da rede de operação (Internet) e dos protocolos utilizados pelo serviço, como *SOAP* e *HTTP*. O *SOAP* emprega o *XML* como formato de dados, o que acarreta um aumento no volume dos dados. Além disso, a manipulação dos dados (tradução das informações *XML* e extração dos dados *SOAP*) implica em esforço de processamento e aumento do consumo de tempo;

7. Modelo de cobrança: serviços Web comerciais necessitarão de um modelo segundo o qual devem ser remunerados pelos serviços prestados;

8. Disponibilidade: para estar disponível para uso, os Serviços Web dependem da disponibilidade do provedor e da própria Internet;

9. Mudanças de interface: a adição de novos métodos ou facilidades a métodos antigos não devem provocar falhas nas aplicações clientes já existentes.

3.4 Processo de Desenvolvimento

O processo de desenvolvimento de Serviços Web pode ser resumido em alguns pontos principais, expostos a seguir.

3.4.1 Codificação

A equipe de desenvolvimento da aplicação realiza a codificação da mesma e a expõe como Serviço Web através de interfaces ou classes bem definidas. Estas interfaces ou classes devem ser capazes de expressar o serviço em termos de parâmetros de entrada (fornecidos pelo cliente) e saída (resposta do serviço ao cliente).

Um exemplo de uma interface simples e bem definida pode ser vista no Código 3.1. Esta interface, escrita em linguagem Java, define um método (somaInteiros) que recebe dois inteiros e retorna a soma dos dois.

```
public interface SomaWS {
        /**
         *
         * @param a número inteiro a
         * @param b número inteiro b
         * @return um inteiro com a soma: a + b
         */
         public int somaInteiros(int a, int b);
}
```

Código 3.1: Interface SomaWS.

Nesta etapa, o descritor do serviço, *WSDL* (Ver seção 4.5), é gerado e, a partir dele, é gerado e implementado o código com a lógica do negócio, baseado no protocolo *SOAP*, que ficará disponível no servidor do Serviço. A este código é dado o nome de *Skeleton*.

A geração do descritor *WSDL* pode ser feita de duas maneiras:

1. A partir de um código escrito em alguma linguagem de programação: existem ferramentas capazes de, a partir da implementação (como a interface mostrada no código 3.1), gerar o documento *WSDL*. Do mesmo modo, estas ferramentas são capazes de ler

documentos *WSDL* e produzir as respectivas implementações em diversas linguagens. O Apache *Axis* (Axis2, 2007) é um exemplo de uma plataforma escrita em Java para criação e consumo de Serviços Web. O *Axis* é discutido mais detalhadamente na seção 4.7.

O Código 3.2 mostra o trecho do descritor *WSDL*, gerado com a ferramenta *AXIS*, onde os métodos e seus tipos são definidos. A seção 4.5 apresenta maiores detalhes sobre o descritor *WSDL*.

```
<wsdl:definitions xmlns:ns="http://exWSBobo"
        ............. ............
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://exWSBobo/xsd">
        <xs:element name="somaInteiros">
          <xs:complexType>
            <xs:sequence>
                <xs:element
                 name="param0" nillable="true" type="xs:int" />
                <xs:element
                  name="param1" nillable="true" type="xs:int" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="somaInteirosResponse">
          <xs:complexType>
            <xs:sequence>
                <xs:element
                  name="return" nillable="true" type="xs:int" />
                </xs:sequence>
          </xs:complexType>
        </xs:element>
   </xs:schema>
 </wsdl:types>
        ............. ............
</wsdl:definitions>
```

Código 3.2: Trecho do *WSDL* gerado para a Interface SomaWS.

2. Escrevendo-se diretamente o arquivo: utilizando a linguagem *XML*, o descritor pode ser

criado com o auxílio de qualquer editor de texto. É possível, também, utilizar programas específicos para edição de *WSDL*, disponíveis na Internet.

A Figura 3.1 ilustra a geração do descritor *WSDL* e do *Skeleton*.

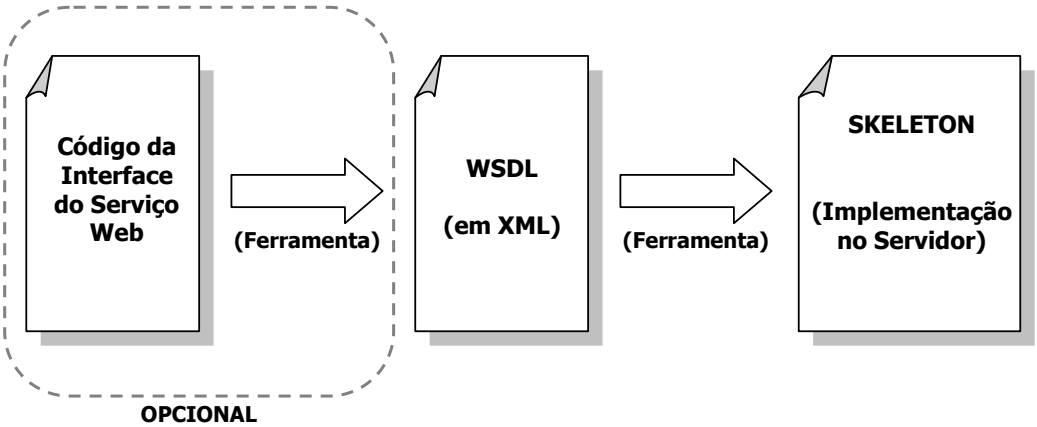


Figura 3.1: Geração do *WSDL* e do *Skeleton*.

O código do *Skeleton* é, então, completado com a lógica do negócio (Serviço Web).

3.4.2 Disponibilização

Nesta etapa, a implementação do Serviço Web é disponibilizada em um contêiner de serviço ou através de um ambiente de execução *SOAP* existentes em um servidor conectado à Internet. Este procedimento também é conhecido como "*Deploy*" (ver Figura 3.2).

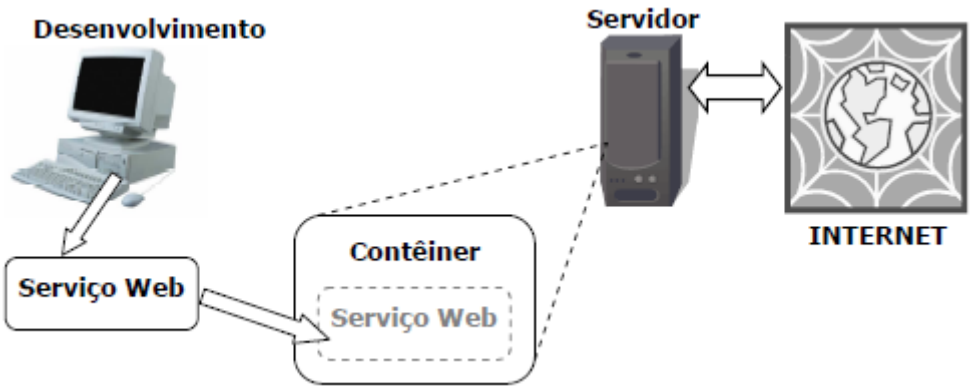


Figura 3.2: *Deploy* do Serviço Web.

O serviço, então, se torna acessível para consumo por um programa cliente.

3.4.3 Operação

Nesta fase inicia-se a operação do Serviço Web, também denominada de "ciclo de vida". Conforme ilustra a Figura 3.3, tal ciclo ocorre em quatro fases:

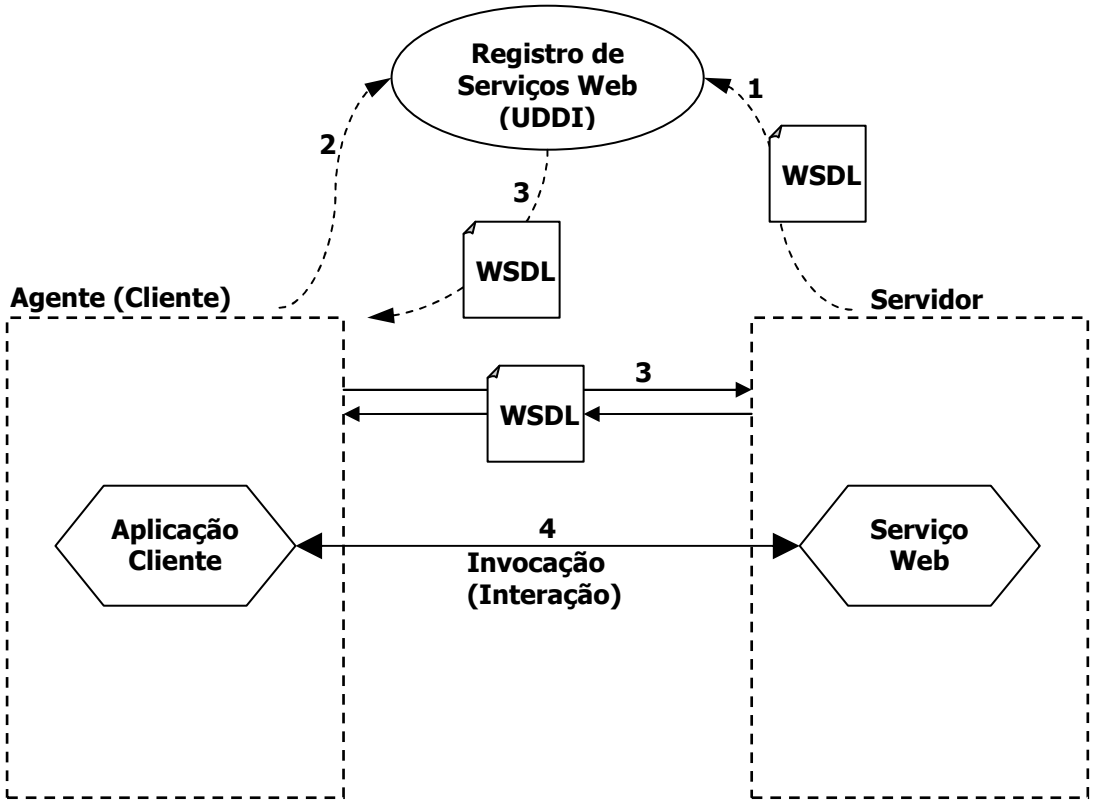


Figura 3.3: Esquema do ciclo de vida de um Serviço Web.

1. **Publicação:** Nesta fase é feito o registro do serviço em um repositório de Serviços Web, *UDDI* (Ver seção 4.6). Esta fase é opcional.

2. **Descoberta:** Processo através do qual uma aplicação cliente toma conhecimento da existência do Serviço Web, pesquisando em um repositório *UDDI*. Esta fase também é opcional.

3. **Descrição:** Processo pelo qual o Serviço Web expõe suas características através do documento *WSDL*, ao público. A aplicação cliente tem, então, acesso à interface do serviço. O próprio repositório de serviços pode fornecer o documento *WSDL*. Caso não o tenha, o repositório fornecerá uma referência a este arquivo (endereço Web ou alguma outra

forma de informação sobre como obter o *WSDL* do serviço). Com base neste descritor, o cliente cria um *proxy* (implementação que encapsula os procedimentos para conexão com o Serviço Web) para realizar a comunicação com o provedor através do protocolo *SOAP*. A geração deste *proxy*, o qual é comumente chamado de "*Stub*", está ilustrada na Figura 3.4.

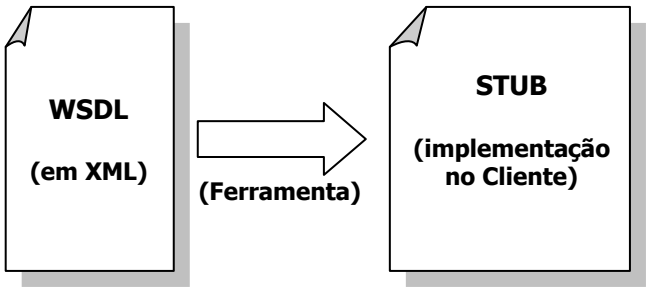


Figura 3.4: Geração do *Stub*.

4. **Invocação:** Processo pelo qual o cliente e o servidor interagem, através de trocas de mensagens *SOAP*.

3.5 Exemplos de Serviços Web

3.5.1 Um Exemplo Simples

A Figura 3.5 ilustra um exemplo de utilização de Serviços Web.

Neste exemplo, um usuário utiliza uma aplicação cliente, no caso um navegador Web acessando o *site* de uma empresa que realiza cotações. Ele preenche um formulário com os dados da pesquisa que são, então, enviados ao servidor, onde a aplicação Web da empresa recebe a requisição. A aplicação localiza os serviços disponíveis (Serviços Web), com base em uma lista pré-estabelecida e envia as mensagens (*SOAP*) contendo as requisições de informações.

Cada Serviço Web das lojas virtuais, recebe e processa o pedido e envia a resposta de volta à aplicação Web solicitante. Esta aplicação recebe e processa as informações recebidas, e envia uma página Web contendo os resultados de volta à aplicação cliente (navegador) (Cunha, 2002).

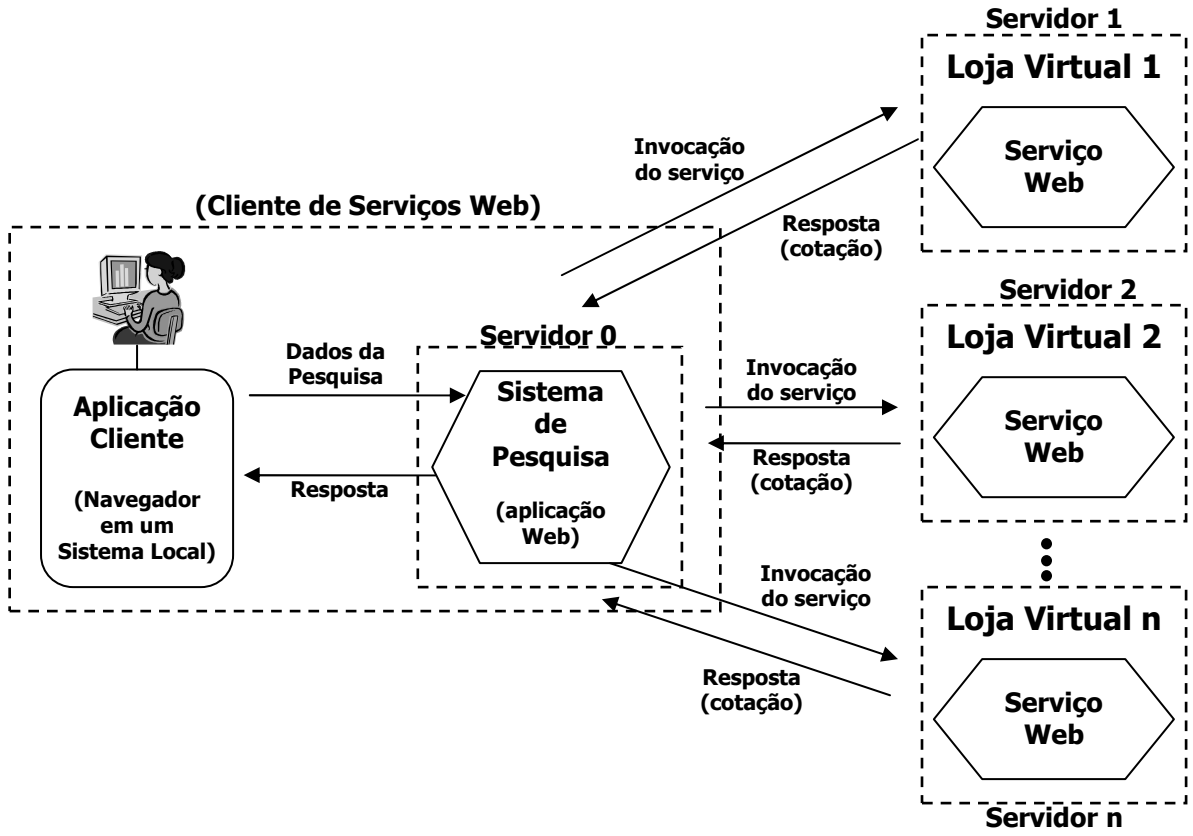


Figura 3.5: Exemplo de utilização de Serviços Web (Cunha, 2002).

3.5.2 Serviços Web Disponíveis

Uma lista de Serviços Web públicos disponíveis para uso pode ser encontrada no *Site* da X Methods `http://www.xmethods.net/` (acessado em julho de 2007). Nesta página, para cada serviço é possível obter o seu descritor *WSDL*, além de outras informações sobre o mesmo.

A Google oferece seu serviço de busca como Serviço Web e é de uso gratuito. Seu descritor pode ser obtido no *site*: `http://api.google.com/GoogleSearch.wsdl`.

3.5.3 Serviços Web Relacionados a este Trabalho

Foi realizada uma pesquisa com o objetivo de encontrar trabalhos relacionados ao tema aqui desenvolvido.

No Brasil não foram encontrados trabalhos de desenvolvimento de Serviços Web na área de elementos finitos. Porém, alguns trabalhos encontrados envolviam processamento remoto,

como o trabalho citado na seção 3.5.3.1.

No exterior, vários trabalhos relacionados a Serviços Web e ao Método dos Elementos Finitos foram encontrados, sendo que alguns estão listados a seguir.

3.5.3.1 Unesp-GridGene

Autores: Ivan Rizzo Guilherme, Carlos Norberto Fischer, Maurício Bacci Júnior.

Resumo: A proposta deste trabalho é desenvolver, utilizando o *framework* do *myGrid*, um ambiente voltado para a construção de uma infra-estrutura de trabalho para a área de Bioinformática, requerido por pesquisadores da UNESP. O "*MyGrid*" é um plataforma estrangeira baseada em Serviços Web (`http://www.mygrid.org.uk/`).

Endereço Web da Publicação:

`http://hep.ift.unesp.br/SPRACE/files/ProjetoGridUNESP.pdf`

(Consulta em 15/07/2007).

3.5.3.2 Building Problem-Solving Environments with Application Web Service Toolkits

Autores: Choonhan Youn, Marlon E. Pierce, and Geoffrey C. Fox (*Syracuse University, Syracuse, NY,* e *Indiana University - USA*).

Resumo: Neste trabalho é levantado o problema do reuso através da apresentação de um conjunto de serviços construídos com o modelo de Serviços Web e serviços de metadados de aplicações que podem ser utilizados na construção de aplicações científicas e no gerenciamento de múltiplas versões de serviços.

Endereço Web da Publicação:

`http://linkinghub.elsevier.com/retrieve/pii/S0167739X03002863`

(Consulta em 15/07/2007).

3.5.3.3 Integration of Chargeable Web Services into Engineering Applications

Autores: Molinari, Marc, Kammuni, Kushan and Cox, Simon J.

Resumo: Discute-se um caso de estudo de engenharia dirigido a questões de integração e segurança de consumo de um pacote de software específico para malhas de elementos finitos através de uma interface baseada em Serviços Web.

Entretanto, o Serviço Web requer cadastro prévio e pagamento pela sua utilização.

Endereço Web da Publicação: `http://eprints.soton.ac.uk/45808/`

(Consulta em 15/07/2007).

3.5.3.4 Internet-Enabled Distributed Engineering (Web) Services

Autores: J. Peng, D. Liu, J. Cheng, C.S. Han and KH. Law - *Stanford University, Stanford, CA, USA*.

Resumo: O artigo apresenta os conceitos básicos da tecnologia de Serviços Web e suas potenciais aplicações na Engenharia Civil. Três exemplos de aplicações são apresentadas para demonstrar que essa abordagem é uma promessa de paradigma para integração de vários *softwares* de engenharia.

Entretanto, não há menção de Serviços Web disponíveis e em funcionamento.

Endereço Web da Publicação:

`http://eil.stanford.edu/publications/jun_peng/Law_Internet_Web_Services.pdf`

(Consulta em 15/07/2007).

3.5.3.5 Computational Science Simulations Based on Web Services

Autores: Paul Chew, Nikos Chrisochoides, S. Gopalsamy, Gerd Heber, Tony Ingraffea, Edward Luke, Joaquim Neto, Keshav Pingali, Alan Shih, Bharat Soni, Paul Stodghill, David Thompson, Steve Vavasis, Paul Wawrzynek (*Cornell University, Mississippi State University* e *University of Alabama at Birmingham*).

Resumo: O artigo descreve uma arquitetura de software de um sistema para realização de simulação "*multiphysics*" de um problema de fratura mecânica, térmico e fluído. O sistema é organizado como uma coleção de componentes de software distribuídos geograficamente, na qual cada componente provê um Serviço Web e utiliza protocolos padrões de Serviços Web

para interação com outros componentes.

Entretanto, não foram indicados os descritores dos Serviços Web mencionados.

Endereço Web da Publicação:

`http://iss.ices.utexas.edu/Publications/Papers/ICCS2003.pdf`

(Consulta em 15/07/2007).

3.5.3.6 NEESgrid: A Distributed Collaboratory For Advanced Earthquake Engineering Experiment And Simulation

Autores: Billie Spencer Jr., Thomas A. Finholt, Ian Foster, Carl Kesselman, Cristina Beldica, Joe Futrelle, Sridhar Gullapalli, Paul Hubbard, Lee Liming, Doru Marcusiu, Laura Pearlman, Charles Severance,Guangqiang Yang (*USA*).

Resumo: NEESgrid, o componente de integração de sistema do projeto NEES, conecta pesquisadores de terremotos pelos Estados Unidos, permitindo equipes colaboradoras (mesmo remotas) planejar, executar e publicar seus experimentos. As ferramentas e componentes disponíveis pelo NEESgrid permitem simulações de engenharia de terremotos, físicas e numéricas, oferecendo um ambiente para pesquisadores desenvolverem modelos precisos e complexos de estruturas de todos os tipos, submetidas a carregamentos de terremotos.

Entretanto, o artigo cita como produto um pacote de software que utiliza, dentre outras aplicações, Serviços Web, os quais não estão detalhados.

Endereço Web da Publicação: `http://www.globus.org/alliance/publications/papers/13worldconferenceonEarthquakeEngineering-rad8A451.pdf`

(Consulta em 15/07/2007).

3.5.3.7 Distributed, High-performance Earthquake Deformation Analysis and Modelling Facilitated by Discovery Net

Autores: Christian Haselwimmer, Yike Guo, Gareth Morgan, Kyran Mish (*Imperial College*, Londres e *University of Oklahoma*).

Resumo: O projeto "*Discovery Net*" que tem investigado a relação entre macro e micro

escala de processos de deformação de terremotos, desenvolveu e testou com sucesso uma infraestrutura de "*geoinformatics*" para interconectar computacionalmente modelagens e monitoramentos intensivos de terremotos. O artigo descreve detalhes do projeto, resultados iniciais de testes de fluxo e pesquisas futuras.

O *Discovery Net* é uma plataforma desenvolvida no *Imperial College London* e é baseada em infraestrutura computacional baseada em serviços. Esta infraestrutura foi estendida para integrar serviços de pós-processamento e análise por elementos finitos. O produto foi disponibilizado como Serviço Web para ser utilizado pela comunidade científica. Não há maiores detalhes da implementação dos Serviços Web, nem como utilizá-la.

Endereço Web da Publicação:

`http://www.allhands.org.uk/2006/proceedings/papers/683.pdf`

(Consulta em 15/07/2007).

Capítulo 4

TECNOLOGIAS RELACIONADAS A SERVIÇOS WEB

4.1 Introdução

O desenvolvimento de Serviços Web, requer o conhecimento e domínio de várias tecnologias, como linguagens de programação, protocolos, entre outras.

Uma linguagem de programação define regras sintáticas e semânticas específicas para um recurso computacional.

Um protocolo, segundo sua definição, é a padronização de leis e procedimentos para a execução de uma determinada tarefa. Na informática, é o termo usado para um conjunto de informações ou dados que passam por um preparo para serem repassados a outros programas. Um protocolo pode definir o uso de determinadas linguagens de programação.

Outras tecnologias oferecem facilidades, ferramentas ou encapsulamento de soluções de mais baixo nível.

Neste capítulo, são apresentadas as principais tecnologias relacionadas aos Serviços Web necessárias ao desenvolvimento deste trabalho de dissertação. Na seção 4.2 é apresentado o protocolo *HTTP* o qual permite aos Serviços Web e navegadores enviar e receber dados pela Internet. A seção 4.3 trata da *XML*, uma linguagem de marcação muito usada em várias outras tecnologias para armazenar e descrever dados ou informações. A seguir, na seção 4.4, é exposto o protocolo *SOAP*, o qual padroniza a informação de interação (mensagens) entre os Serviços Web e aplicações cliente. A seção 4.5 apresenta a linguagem *WSDL* que estabelece um modelo

e um formato *XML* para descrever completamente um Serviço Web (sua interface, os tipos de dados utilizados e o seu endereço na Internet). A seção 4.6 expõe o padrão *UDDI* baseado em *XML* e utilizado para registros de informações sobre Serviços Web existentes na Internet. Finalmente, a seção 4.7 apresenta a ferramenta *Axis* 2 que é uma implementação do protocolo *SOAP* e oferece uma arquitetura modular e um modelo de objetos completo que torna simples o emprego de especificações e recomendações relacionadas aos Serviços Web.

É importante salientar, que algumas das tecnologias expostas neste capítulo também podem ser ou são utilizadas em aplicações Web (discutidas no Capítulo 5), tais como o *HTTP*, o *XML* e o *Axis*. Entretanto, elas são apresentadas neste capítulo por serem imprescindíveis em Serviços Web ou, como no caso do *Axis*, por apresentarem recursos utilizados em ambas as situações.

4.2 Hypertext Transfer Protocol (HTTP)

O *HTTP* é o protocolo que permite aos Serviços Web e navegadores enviar e receber dados pela Internet (Kurniawan e Deck, 2004). É um protocolo de requisição (*Request*) e resposta (*Response*). O cliente (o navegador) faz uma requisição ao servidor Web. O servidor processa a requisição e envia a resposta ao navegador (ver Figura 4.1). Esta comunicação é feita através de mensagens *HTTP*.

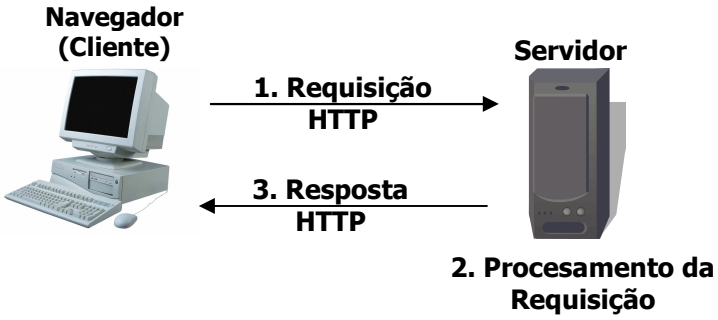


Figura 4.1: Fluxo de uma requisição *HTTP*.

O *HTTP* utiliza conexões confiáveis TCP (*Transport Control Protocol* ou Protocolo de Controle de Transmissão). Este protocolo cuida da integridade e confiabilidade da transmissão dos dados (garante a entrega ao seu destino).

A Figura 4.2 ilustra um exemplo de mensagem de requisição *HTTP* e a Figura 4.3 mostra um exemplo de resposta. A composição e o formato destas mensagens são explicadas mais detalhadamente no Apêndice G, seção G.2.

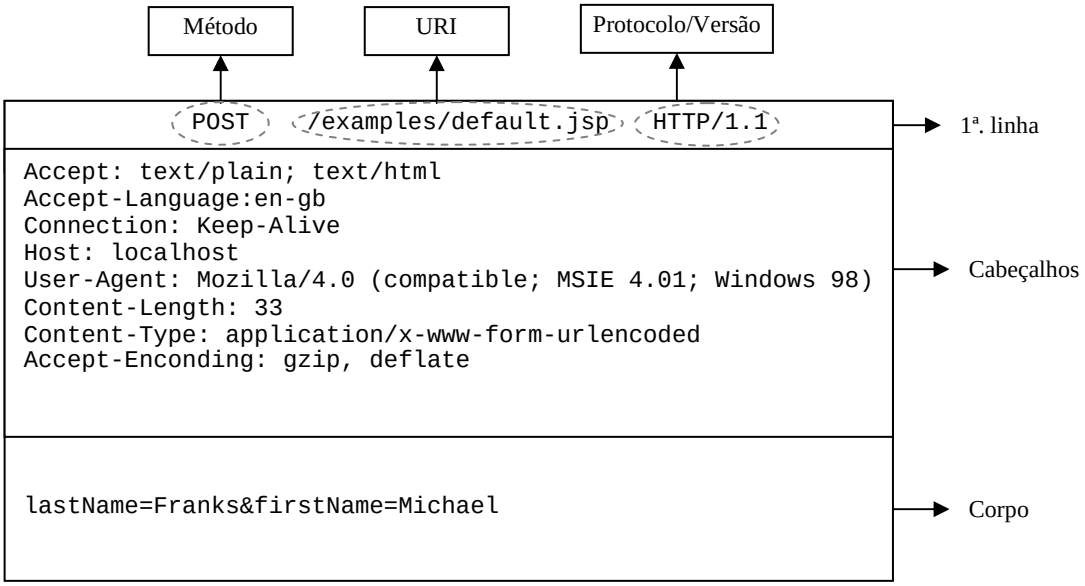


Figura 4.2: Exemplo de uma requisição *HTTP* (Kurniawan e Deck, 2004).

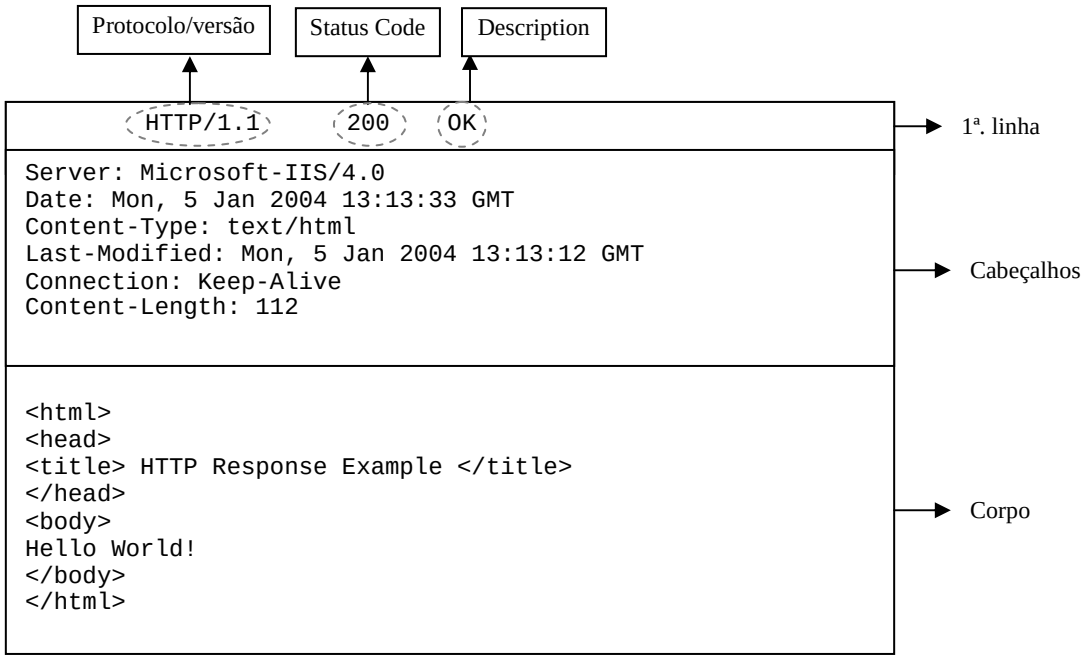


Figura 4.3: Exemplo de uma resposta *HTTP* (Kurniawan e Deck, 2004).

4.3 XML

XML significa "*Extensible Markup Language*" ou, em português, Linguagem de Marcação Extensível.

Entretanto, a *XML* não é apenas uma linguagem de marcação, mas um conjunto de regras para a criação de linguagens de marcação. É um conjunto de ferramentas para armazenamento de dados, um veículo configurável para qualquer tipo de informação e um padrão aberto e em evolução, abraçado por todos, desde banqueiros até "*webmasters*" (Ray, 2001).

Enquanto a *HTML* é usada para formatação e exibição de informações, a *XML* é usada para descrever e armazenar essas informações.

Assim como a *HTML*, a *XML* usa marcas e atributos para formatação de elementos, mas, por ser extensível, a *XML* permite a criação de elementos. As marcas *XML* definem uma estrutura para a informação.

As principais características da *XML* são (Ray, 2001):

- A *XML* pode armazenar e organizar praticamente qualquer tipo de informação em um formato adequado às necessidades do usuário;
- Como um padrão aberto, a *XML* não está ligada às fortunas de qualquer empresa isolada, e nem acoplada a qualquer *software* em particular;
- Com o Unicode como seu conjunto de caracteres padrão, a *XML* aceita um número muito grande de sistemas escritos e símbolos, desde caracteres rúnicos escandinavos até ideógrafos chineses;
- A *XML* oferece muitas maneiras de verificar a qualidade de um documento, com regras para sintaxe, verificação de vínculo interno, comparação com modelos de documento e tipos de dados;
- Com sua sintaxe clara e simples e sua estrutura sem ambigüidades, a *XML* é fácil de ler e analisar por seres humanos e por programas de computador;

– A *XML* é facilmente combinada com folhas de estilo para criar documentos formatados em qualquer estilo que se queira. A pureza da estrutura da informação não atrapalha as conversões de formato.

A estrutura de um documento *XML* é composta por:

– Declaração *XML* (obrigatória);

– Declaração do tipo de documento;

– Instruções de processamento;

– Comentários;

– Texto (dados);

– Elementos e seus atributos;

– Definição do tipo de documento: DTD ou *XML* *Schema* (ver apêndice G.3.5).

Os elementos dividem o documento em suas partes constituintes. Eles podem conter texto, outros elementos ou ambos. Esta estrutura define uma hierarquia facilmente visualizada na codificação *XML*. O elemento raiz ou principal deve ser único e deve conter todos os outros elementos do documento.

O Código 4.1 mostra o conteúdo do documento de exemplo "`book.xml`" e a hierarquia formada pelo aninhamento dos seus elementos pode ser visualizada na Figura 4.4.

```
<?xml version="1.0"?> <!DOCTYPE book
    PUBLIC "-//ORA//DTD DBLITE XML/EN"
    SYSTEM "/usr/local/prod/dtds/dblite.dtd"
[
    <!ENTITY chap1 SYSTEM "ch01.xml">
    <!ENTITY chap2 SYSTEM "ch02.xml">
    <!ENTITY xml "<acronym>XML</acronym>">
]>
<book>
  <title>User Manual</title>
  <author>Indigo Riceway</author>
  <preface id="preface">
     <title>Preface</title>
     <sect1 id="about">
       <title>Availability</title>
<!-- Nota para o autor: talvez colocar uma figura aqui: -->
       <para>
         The information in this manual is available in the following
         forms:
       </para>
       <itemizedlist>
          <listitem> Instant telepathic injection </listitem>
          <listitem> Lumino-google display </listitem>
                ....
          </itemizedlist>
    </sect1>
  </preface>
  <chapter id="intro">
     <title>Introduction</title>
     <para>
            Congratulations on your purchase of one of the most
            ......
     </para>
  </chapter>
</book>
```

Código 4.1: `book.xml` - adaptado de Ray (2001).

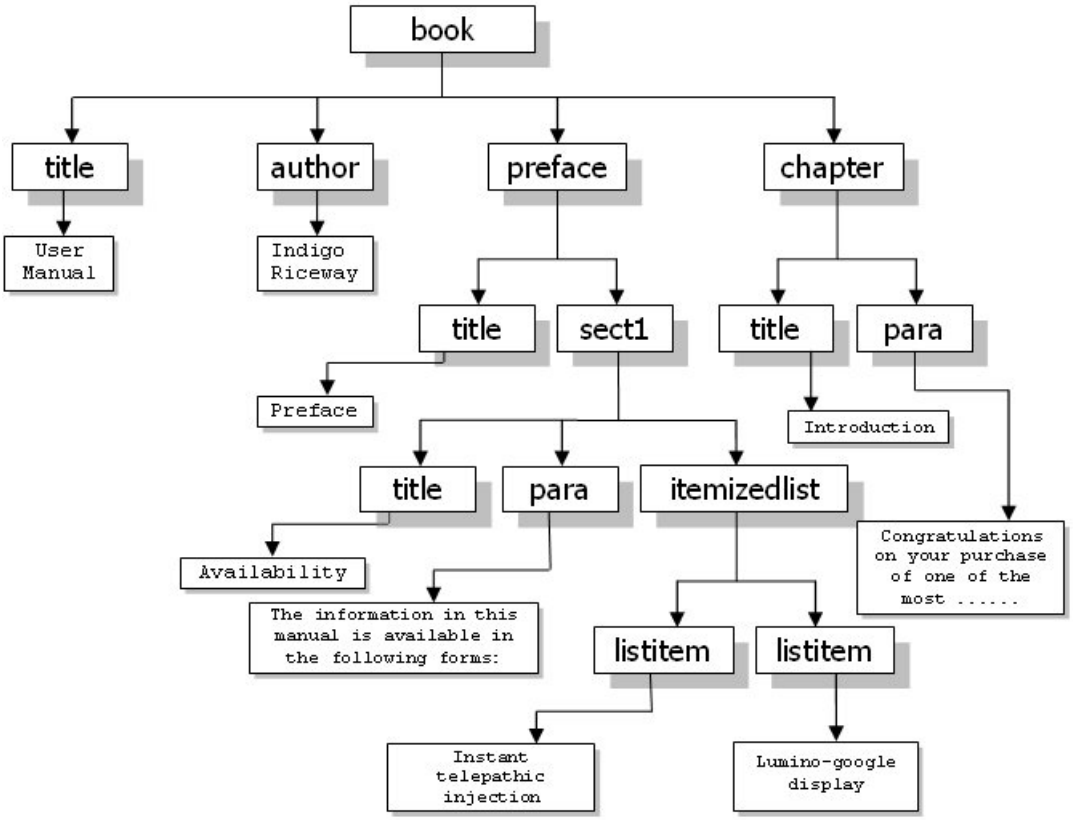


Figura 4.4: Hierarquia de elementos *XML* do Código 4.1.

A *XML* permite a criação livre de elementos e atributos. É possível definir formalmente uma linguagem em *XML* através de um processo chamado *modelagem de documentos*. Através desta modelagem, pode-se restringir o vocabulário dos elementos e atributos e controlar a gramática dos elementos.

O modelo é um tipo especial de documento, escrito em uma sintaxe criada para descrever linguagens *XML*, que estabelece explicitamente o vocabulário para uma única linguagem de marcação.

O tipo mais popular de modelo de documento é a DTD (*Document Type Definition*), que é um arquivo texto composto de uma seqüência de declarações.

Outro tipo de modelo de documento é o *XML Schema*. Ao contrário da DTD, ele é escrito em *XML* e oferece muito mais controle sobre os tipos e padrões de dados, tornando-o uma linguagem mais atraente para impor requisitos escritos de entrada de dados.

O Apêndice G, seção G.3 apresenta maiores detalhes sobre a estrutura e sintaxe usadas na

linguagem *XML*, assim como maiores informações sobre seus modelos de documento.

4.4 SOAP

O *SOAP* (*Simple Object Access Protocol*), também referido como "*Service-Oriented Architecture Protocol*" ou, em português, Protocolo de Arquitetura Orientada a Serviços, é um dos mais conhecidos formatos de mensagens e protocolos utilizado por Serviços Web baseados em *XML*. Uma mensagem *SOAP* é um arquivo *XML* que carrega de forma padronizada toda a informação da interação entre Serviços Web e aplicações cliente (Erl, 2005).

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 de programação.

Normalmente, mensagens *SOAP* são enviadas utilizando o *HTTP* como protocolo de transmissão de dados. Entretanto, *SOAP* é independente do protocolo de transporte, podendo também utilizar outros protocolos como SMTP (*Simple Mail Transfer Protocol*), "*rwa sockets*", entre outros (Albinader e Lins, 2006). A Figura 4.5 ilustra a estrutura de uma mensagem *SOAP* e seu relacionamento com o protocolo *HTTP*.

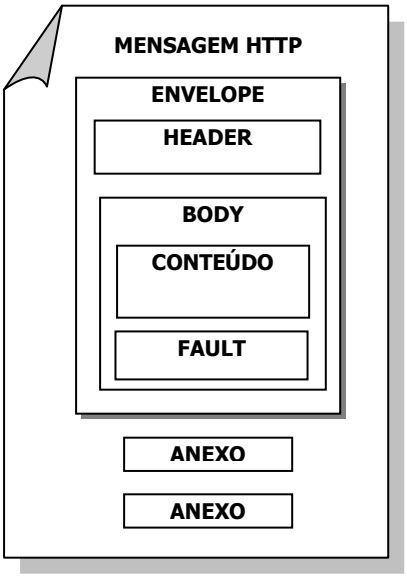


Figura 4.5: Estrutura de uma mensagem *SOAP*. Adaptada de Albinader e Lins (2006) e Cunha (2002).

Conforme mostra a Figura 4.5, a mensagem *SOAP* é formada pelos seguintes elementos:

- **Envelope**: é o elemento raiz do documento *XML*. A principal função deste elemento é indicar ao receptor onde começa e termina a mensagem;
- **Header**: é um cabeçalho opcional. Ele carrega informações adicionais, como por exemplo, informações de processamento em nós intermediários (ao trafegar pela rede, a mensagem normalmente passa por vários pontos intermediários até alcançar seu destino), informações de segurança e outros. Quando utilizado, o *Header* deve ser o primeiro elemento do envelope;
- **Body**: é um elemento obrigatório, dentro do qual é disposto o conteúdo para o receptor final da mensagem. O elemento *Body* pode conter um elemento opcional denominado *fault*, que é usado para armazenar mensagens de status e erros retornados pelos nós de processamento.

O Código 4.2 mostra um exemplo de mensagem *SOAP*, na qual um cliente de um Serviço Web solicita informações sobre um produto.

O Código 4.3 mostra a resposta do Serviço Web a esta solicitação.

Os exemplos foram retirados do "*site*": `http://www.w3schools.com/soap/soap_example.asp` em 12/10/06.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

Código 4.2: Exemplo de mensagem *SOAP* para solicitação de informações de um produto.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

Código 4.3: Mensagem *SOAP* de resposta para a solicitação descrita no Código 4.2.

4.5 WSDL

A *WSDL* (*Web Service Description Language*) é uma linguagem que estabelece um modelo e um formato *XML* para descrever um Serviço Web.

Um documento *WSDL* é um arquivo em formato *XML* padronizado que descreve o serviço remoto de maneira estruturada. Ele descreve a interface do serviço, os tipos de dados usados e onde o serviço está localizado.

A especificação *WSDL* define a estrutura de um documento *WSDL* a qual possui seis principais elementos representados na Figura 4.6 e discutidos a seguir:

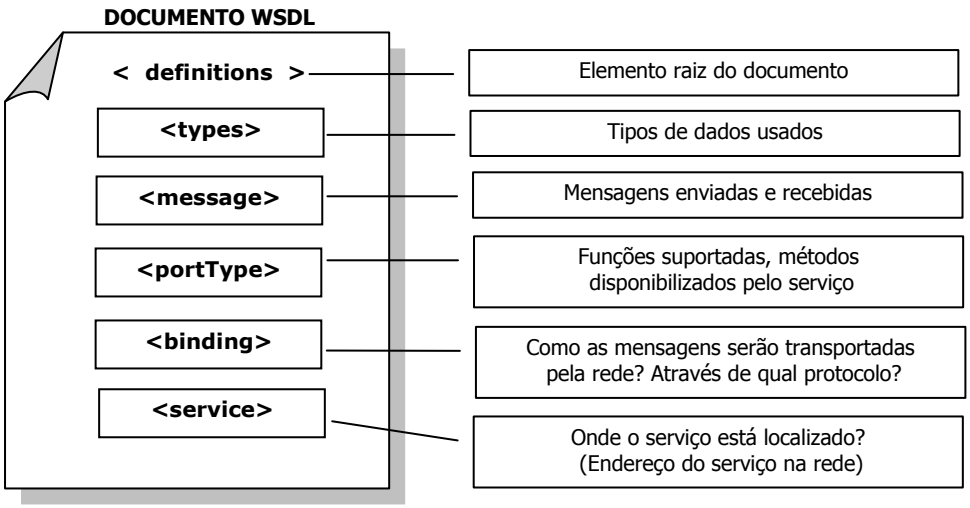


Figura 4.6: Estrutura de um documento *WSDL*.

1. Elemento **<definitions>**: é o elemento raiz do documento *WSDL* e define o nome do Serviço Web;

2. Elemento **<types>**: especifica e lista todos os tipos usados pelo Serviço Web. A especificação *WSDL* herda o conjunto de tipos padrão do *XML Schema*. Assim, se o serviço não usa nenhum tipo além dos definidos no *XML Schema*, o elemento <types> não é necessário;

3. Elemento **<message>**: todas as mensagens que possam vir a ser trocadas entre o consumidor do serviço e o serviço são listadas no documento *WSDL* através de elementos <message>.

4. Elemento **<portType>**: é a interface abstrata do Serviço Web. É um conjunto de operações e mensagens abstratas envolvidas (Christensen et al., 2001).

5. Elemento **<binding>**: associa protocolos específicos para a invocação do serviço aos elementos definidos no elemento <portType>.

6. Elemento **<service>**: define o endereço de rede do serviço a ser invocado.

O Código 4.4 mostra um exemplo de arquivo *WSDL* e no Apêndice G, seção G.4 são apresentados maiores detalhes sobre os elementos *WSDL*.

Através do documento *WSDL* de um Serviço Web, é possível construir uma aplicação cliente para este serviço, pois ele contém todas as informações necessárias para a sua utilização. Na seção 4.7 é apresentada uma ferramenta chamada *Axis*, que gera código Java a partir do arquivo *WSDL* e vice-versa.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
   targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>
   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>
   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
         <soap:operation soapAction="sayHello"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </input>
         <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </output>
      </operation>
   </binding>
   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address
            location="http://localhost:8080/soap/servlet/rpcrouter"/>
      </port>
   </service>
</definitions>
```

Código 4.4: Exemplo de um arquivo *WSDL* (Cerami, 2002).

4.6 UDDI

UDDI ou "*Universal Description, Discovery and Integration*" é um padrão para registros baseados em *XML* que contêm informações sobre um Serviço Web. Um registro *UDDI* permite o acesso aos documentos *WSDL* dos Serviços Web existentes na Internet (que foram publicados). As empresas ou qualquer um que queira divulgar seu Serviço Web faz a publicação das suas informações através de um registro *UDDI* e qualquer um que queira usar este serviço pode encontrar as suas informações através de uma busca em um repositório de registros *UDDI* (Bellwood, 2002).

Um registro *UDDI* contém informações sobre negócios, dados da organização e dos serviços que ela oferece. Ele pode ser comparado a uma lista telefônica dividida em três grandes partes:

1. **Páginas brancas**: permitem encontrar informações sobre a empresa, como endereço, informações de contato com a empresa, etc.

2. **Páginas amarelas**: as informações são classificadas com base em padrões da indústria ou segmento da qual a empresa e os seus serviços fazem parte.

3. **Páginas verdes**: contém a lista completa dos serviços na Internet oferecidos pelas empresas relacionadas nas páginas brancas.

Com a finalidade de permitir a busca e descoberta de organizações e dos serviços oferecidos por elas, através da abordagem solicitação-resposta, o *UDDI* disponibiliza *APIs* (*Application Program Interface*) que tornam estas funções executáveis através de programas.

Estão disponíveis dois tipos de *API* para interação via programa (Albinader e Lins, 2006):

1. ***API* de publicação**: é empregada para interação entre aplicações do publicador e o registro *UDDI*, permitindo a criação, modificação e exclusão de registros.

2. ***API* de busca e localização**: destinada a permitir que aplicações realizem buscas, pesquisas e localizem registros.

4.7 Axis

O *Axis* 2, um projeto da Apache (`http://www.apache.org`), é uma implementação do protocolo *SOAP* (ver seção 4.4) utilizado em Serviços Web. Ele oferece uma arquitetura modular e um modelo de objetos completo que torna simples o emprego de especificações e recomendações relacionadas aos Serviços Web (Axis2, 2007).

O *Axis* 2 permite realizar as seguintes tarefas:

1. Envio de mensagens *SOAP*;
2. Recebimento e processamento de mensagens *SOAP*;
3. Criação de Serviços Web a partir de classes Java;
4. Criação de implementações (por exemplo, classes Java) para aplicações clientes e servidoras a partir do descritor *WSDL*;
5. Fácil obtenção do *WSDL* para um serviço;
6. Envio e recebimento de mensagens *SOAP* com anexos;
7. Criação ou utilização de Serviços Web baseados em REST ou *Representational State Transfer* (Costello, 2007);
8. Criação ou utilização de serviços que utilizem as recomendações de Serviços Web em termos de segurança, confiabilidade, endereçamento, entre outras;
9. Uso da sua estrutura modular para facilmente suportar novas recomendações.

Atualmente, o *Axis* 2 é disponibilizado para as linguagens Java e C++ (Axis2, 2007).

4.7.1 Funcionamento

A Figura 4.7 demonstra o funcionamento do *Axis* em um sistema em que o *Axis* esteja sendo usado tanto pelo cliente quanto pelo servidor do Serviço Web (o que não é obrigatório) (Axis2, 2007):

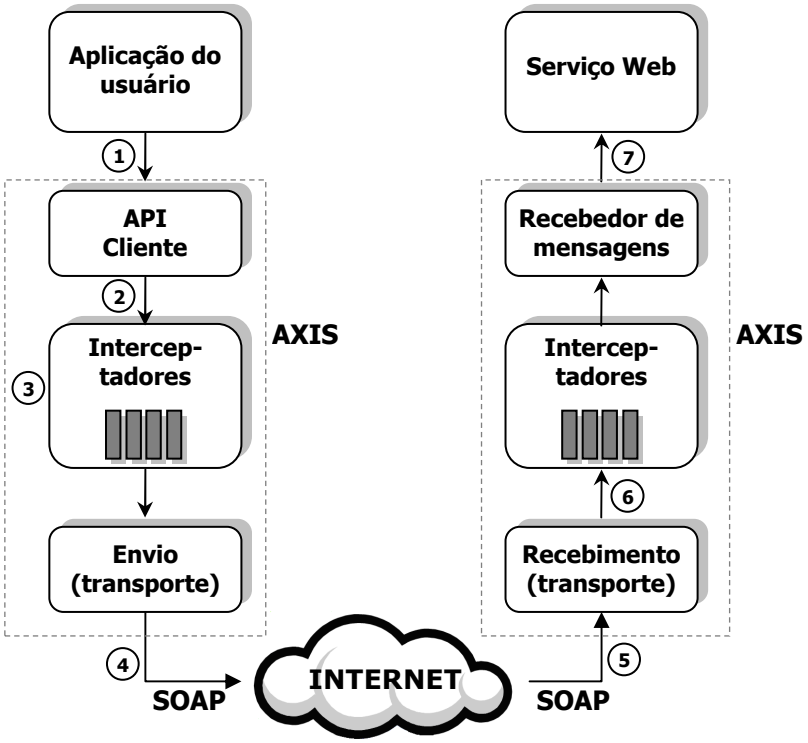


Figura 4.7: Esquema de funcionamento do *Axis*.

1. A aplicação cliente invoca o uso do Serviço Web fazendo uma chamada à *API* cliente (implementação do *Axis*);

2. A mensagem *SOAP* é criada e enviada aos interceptadores;

3. Caso seja necessário, ações como criptografia de segurança são realizadas sobre a mensagem;

4. A mensagem é enviada pela camada de transporte;

5. Na outra ponta (passando pela Internet) a mensagem é recebida por um ouvinte da camada de transporte que a detecta;

6. A mensagem é repassada aos interceptadores para possíveis manipulações;

7. A mensagem é encaminhada à aplicação apropriada (Serviço Web).

A plataforma *Axis* é formada por vários componentes, cada um responsável por uma destas fases de operação.

4.7.2 Disponibilização de Serviços Web através do Axis

Através do *Axis*, é possível a disponibilização dos Serviços Web, de maneira rápida e fácil. No lado do servidor, o *Axis* é instalado em um contêiner Web como o *Tomcat* (ver seção 5.5), funcionando como um *Servlet*. Serviços Web apropriadamente "empacotados" podem, então, ser instalados no *Axis*, como ilustra a Figura 4.8.

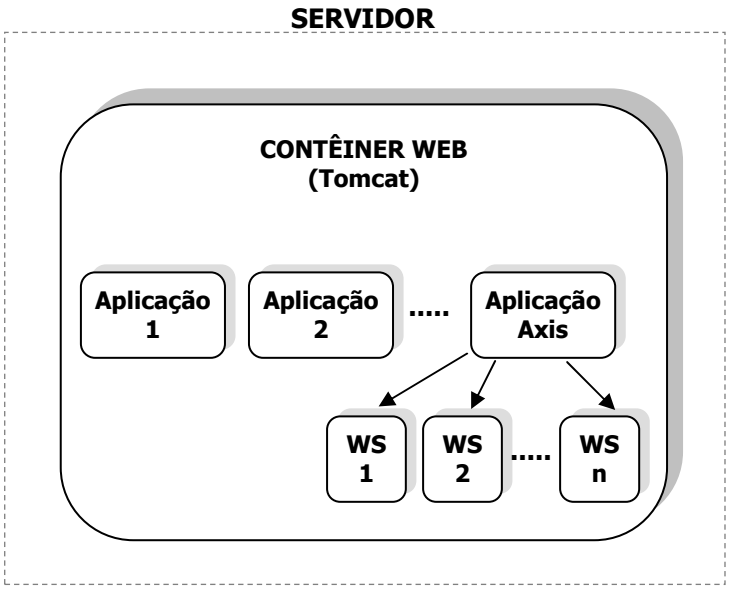


Figura 4.8: Disponibilização de Serviços Web com o *Axis*.

Na versão corrente na época do desenvolvimento deste trabalho (2007), o *Axis*2 versão 1.2, os Serviços Web são empacotados em um arquivo de extensão ".aar" (*Axis* archive), similar ao "jar" da linguagem Java, para serem instalados no *Axis*. Este arquivo deve conter, além das classes do Serviço Web (*skeleton*), um arquivo de configuração (`services.xml`) e arquivos jar de terceiros (outras classes necessárias ao serviços, dispostas em arquivos de extensão ".jar").

O arquivo de configuração do serviço, escrito em *XML* e com o nome `services.xml`, serve para informar ao *Axis* as operações disponíveis para o serviço, o nome da sua classe de implementação (*skeleton*), além de outros parâmetros específicos.

O Código 4.5 mostra o arquivo `services.xml` para o Serviço Web de exemplo mostrado na seção 3.4.1.

A instalação dos Serviços Web no *Axis* é feita copiando-se o arquivo ".aar" para o diretório

```
<!-- This file was auto-generated from WSDL -->
<!-- by the Apache Axis2  -->
<serviceGroup>
  <service name="SomaWS">
    <messageReceivers>
        <messageReceiver
           mep="http://www.w3.org/2004/08/wsdl/in-out"
           class="exwsbobo.SomaWSMessageReceiverInOut" />
        </messageReceivers>
        <parameter name="ServiceClass" locked="false">
         exwsbobo.SomaWSSkeleton
        </parameter>
        <operation name="somaInteiros"
            mep="http://www.w3.org/2004/08/wsdl/in-out">
          <actionMapping>urn:somaInteiros</actionMapping>
          <outputActionMapping>
                http://exWSBobo/SomaWSPortType/somaInteirosResponse
          </outputActionMapping>
        </operation>
  </service>
</serviceGroup>
```

Código 4.5: Arquivo `services.xml` para o Serviço Web SomaWS.

"*services*" do ***Axis*** instalado no contêiner Web ou utilizando-se a interface Web acessível pelo endereço `http://remoteserver:nnnn/axis2` (onde "*remoteserver*" deve ser substituído pelo endereço remoto do contêiner Web e "nnnn" é a porta de acesso ao mesmo), escolhendo-se a opção "*administration*" e depois "***upload*** *service*". Com esta última opção é possível fazer facilmente o ***deploy*** de novos Serviços Web remotamente.

Uma das grandes vantagens do ***Axis***2 é o chamado de "*hotdeploy*", ou seja, não é preciso reiniciar o sistema onde o ***deploy*** foi feito. A aplicação ***Axis*** automaticamente irá tornar o serviço disponível.

Os serviços disponíveis através do ***Axis*** podem ser visualizados escolhendo-se a opção "*services*" no endereço da aplicação ***Axis*** citado acima.

4.7.3 Utilitários Axis

O *Axis* oferece uma série de utilitários que facilitam o desenvolvimento dos Serviços Web, como geradores de código a partir de descritores *WSDL* e vice-versa e geradores de arquivos ".aar" para *deploy*.

Estas ferramentas são disponibilizadas como "*plug-in*s" para ambientes de desenvolvimento como o Eclipse (ver Apêndice C) ou para outros utilitários como o Maven (Apêndice C), ou ainda, como aplicações isoladas.

As ferramentas *WSDL2Java*, que gera código Java a partir do arquivo *WSDL* e *Java2WSDL* que gera o documento *WSDL* a partir de uma interface (ou classes) Java são especialmente úteis no desenvolvimento de Serviços Web. O funcionamento destas ferramentas estão apresentadas no Apêndice G, seção G.5.

4.7.4 AXIOM

O *Axis*2 trouxe uma grande inovação, o "*AXIs Object Model*" (*AXIOM*), um processador *XML* poderoso que promete aumentar a eficiência em manipulação de documentos *XML*.

Ao contrário dos modelos de documentos convencionais, como DOM ou JDOM, *AXIOM* constrói a representação do documento *XML* na memória apenas quando ele é acessado e de modo incremental.

A manipulação de *XML* pelo *AXIOM* (leitura e escrita) é feita, efetivamente, utilizando-se o STAX (*Streaming API for XML*).

O *AXIOM* traz ao *Axis* um processamento *XML* mais rápido e de menor consumo de memória, aumentando assim a eficiência na manipulação de arquivos *XML*. Ele também oferece suporte a arquivos binários com os padrões XOP e MTOM (AXIOM, 2007) e, além de tudo isso, o uso da *API AXIOM* é bastante fácil e simples.

Apesar do *Axis*2 ser baseado no *AXIOM*, este é totalmente independente e pode ser obtido separadamente no *site* da Apache (AXIOM, 2007).

Capítulo 5

TECNOLOGIAS RELACIONADAS A APLICAÇÕES WEB

5.1 Introdução

O desenvolvimento de aplicações Web, assim como de Serviços Web, requer o conhecimento e domínio de diversas linguagens de programação, protocolos e outras tecnologias.

É importante ressaltar a diferença entre Serviços Web e Aplicações Web. O primeiro foi discutido no Capítulo 4, e se refere a um programa que é executado (invocado) por outro programa e usa padrões públicos. É executado pela Internet, porém não apresenta interface direta com o usuário (pois foi idealizado para ser utilizado por outros programas).

Por outro lado, uma Aplicação Web se refere a um programa que é executado pela Web, tipicamente através de um navegador Web e oferece interação com o usuário. Normalmente, é composta por imagens, textos, entre outros arquivos estáticos e uma biblioteca de classes (código de programação compilado) usada na execução do programa. Uma aplicação Web pode invocar Serviços Web, mas o contrário não pode ocorrer.

Neste Capítulo, são apresentadas as principais tecnologias necessárias ao desenvolvimento da Aplicação Web criada para ilustrar o uso do Serviço. A seção 5.2 apresenta a linguagem *HTML* utilizada na construção de páginas Web e a seção 5.3 apresenta a *CSS*, uma linguagem utilizada para a formatação destas páginas. A seção 5.4 expõe o *Tapestry*, uma poderosa biblioteca Java que permite gerar conteúdo dinamicamente em aplicações Web. Na seção 5.5 é apresentado o *Tomcat*, um servidor de aplicações Java para Web que permite que estas sejam

executadas pela Internet. Finalmente, a seção 5.6 trata da *XSL*, uma família de recomendações para transformação e apresentação de documentos *XML*.

5.2 HTML

HTML é a sigla para "*HyperText Markup Language*" ou, em português, linguagem de marcação para hipertexto.

Um documento é considerado hipertexto quando contém referências internas (*links* ou *hyperlinks*) para outros documentos, facilitando a distribuição, atualização e pesquisa por informações relevantes. Documentos hipertextos podem ser encontrados em páginas Web, documentos PDF (*Portable Document Format*) ou na ajuda (*HELP*) do *Windows* (Cicconi, 2005).

Marcação é a informação incluída em um documento para melhorar seu significado por identificar as partes e como elas se relacionam umas com as outras. Uma linguagem de marcação é um conjunto de símbolos que podem ser colocados no texto de um documento para demarcar e rotular as partes desse documento.

A marcação é importante para os documentos eletrônicos porque eles são processados por programas de computador. Se um documento não tiver rótulos ou limites, então um programa não saberá como tratar uma parte do texto para distingui-la de qualquer outra parte (Ray, 2001).

A *HTML* é a linguagem usada na construção de páginas Web. Ela define a estrutura de um documento Web através das marcas e seus atributos. As marcas não são visualizadas literalmente, mas interpretadas pelo navegador a fim de produzir algum efeito visual ou de estrutura na página. Algumas marcas não produzem nenhum efeito, mas contém informações úteis ao navegador.

O navegador ou "*browser*" é o programa que visualiza as páginas Web. O *Firefox*, o *Netscape* e o *Internet Explorer*, são exemplos de navegadores.

As marcas são palavras especiais delimitadas pelos caracteres menor (<) e maior (>). Elas podem ser escritas em letras maiúsculas ou minúsculas. <TITLE>, <HEAD>, <P>,
 são

exemplos de marcas *HTML*. As marcas podem delimitar a região onde ela será aplicada. Para isso basta que a marca seja repetida no fim desta região com uma barra indicando seu término de atuação. No exemplo a seguir, a frase "texto qualquer" será formatada em negrito.

```
<b> texto qualquer </b>
```

A estrutura básica de um documento *HTML* (normalmente um arquivo com extensão .htm ou .html) é:

```
<HTML>
    <HEAD>
        Parte reservada para informações sobre o documento
        (título, palavras chaves, etc.)
    </HEAD>
    <BODY>
        Parte do documento que será efetivamente exibida pelo
        navegador.
    </BODY>
</HTML>
```

Existem muitas marcas *HTML* para as mais diversas finalidades. O Apêndice H, seção H.2 apresenta uma tabela com as marcas *HTML* mais usadas.

O Código 5.1 mostra o conteúdo do documento *exemplo.htm* que será usado para ilustrar o funcionamento de uma página *HTML*[1]. Este exemplo utiliza algumas marcas *HTML* para formatar um texto.

A Figura 5.1 mostra esta página visualizada pelo navegador *Firefox*.

[1]É possível visualizar o código de qualquer página *HTML* através da opção "exibir código fonte" do navegador.

```
<HTML>
    <HEAD>
        <title> Página demonstrativa </title>
    </HEAD>
    <BODY bgcolor="#FFFF99">
        <center><h1> Página demonstrativa </h1></center>
        Este é um <i> pequeno </i> exemplo de página Web.
        <ul>
            <li> Meu nome é <b>Luciana</b>.
            <li> Este é um item de uma lista.
        </ul>
    </BODY>
 </HTML>
```

Código 5.1: Arquivo `exemplo.htm`

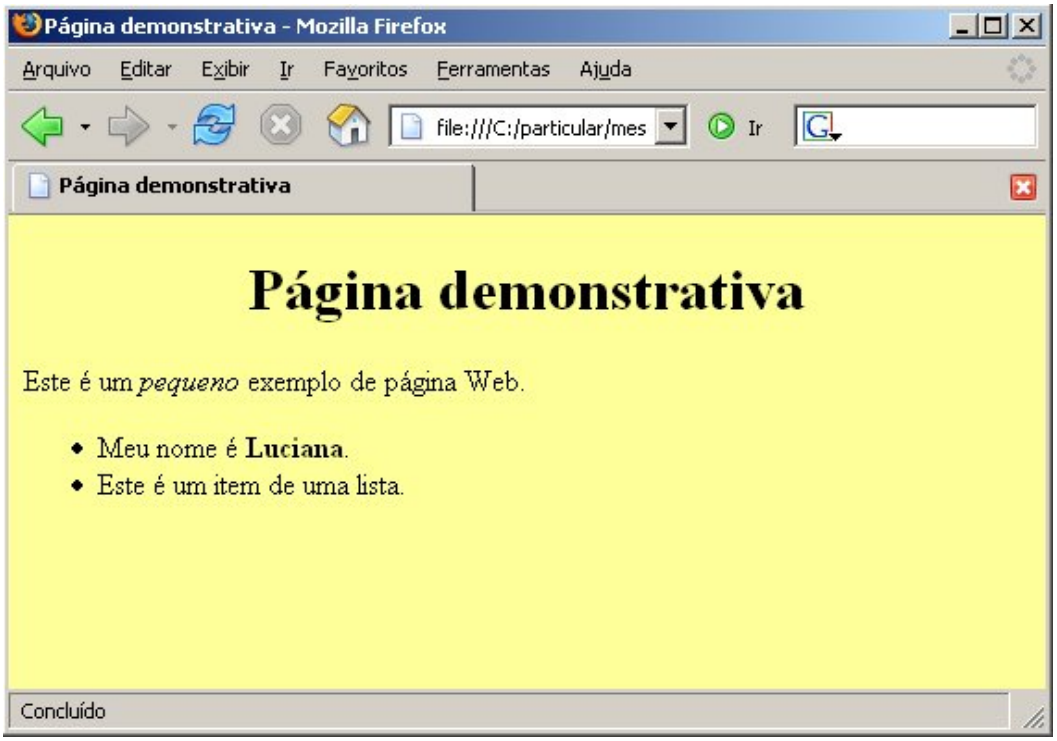


Figura 5.1: Página *HTML* (`exemplo.htm`) visualizada pelo navegador *Firefox*.

5.3 CSS

A *CSS* (*Cascading Style Sheets*) ou folha de estilo em cascata, é uma linguagem para formatação de páginas Web.

Proposta pelo "*W3 Consortium*" (`http://www.w3.org`), uma espécie de comitê que define os padrões de programação para a WWW (*World Wide Web*), a *CSS* foi introduzida pela primeira vez pela *Microsoft*, no lançamento do *Internet Explorer* 3.0.

As folhas de estilo definem como os elementos *HTML* serão visualizados pelo navegador. A Figura 5.2 esquematiza o funcionamento em conjunto de documentos *HTML* e *CSS*.

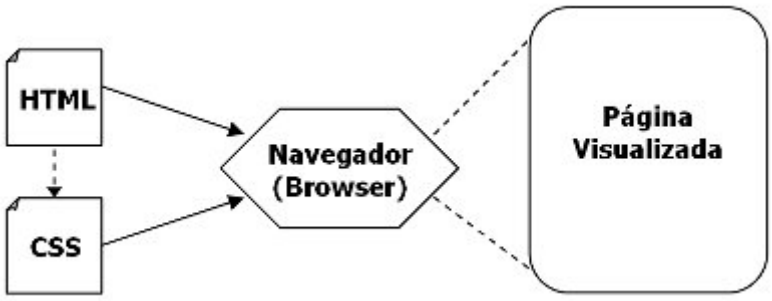


Figura 5.2: Esquema do funcionamento em conjunto de documentos *HTML* e *CSS*.

Basicamente, a *CSS* permite ao desenvolvedor do "*Site*" um controle maior sobre os atributos de uma página, como tamanho e cor das fontes, espaçamento entre linhas e caracteres, margem do texto, caixas de texto, botões de formulário, entre outros. Com a *CSS* tornou-se possível, também, a utilização de "*layers*" (camadas) nas páginas, permitindo a sobreposição de textos e imagens (Silva, 2006).

As regras *CSS* devem ser escritas segundo uma sintaxe própria. Cada regra é composta por três partes: um seletor, uma propriedade e um valor. A sintaxe da regra deve ser:

```
seletor { propriedade: valor; }
```

O seletor é a marca (*tag*) de um elemento *HTML* à qual se deseja atribuir uma formatação. <p>, <h1>, <form> são exemplos de marcas *HTML*.

A propriedade é o atributo do elemento ao qual se aplicará a regra, como por exemplo: *font*, *color* e *size*. Várias propriedades de um mesmo seletor devem ser separadas por um ponto e vírgula.

O valor é a característica a ser aplicada à propriedade, como por exemplo: letra tipo "*arial*", cor da fonte azul, cor do fundo branco, entre outras.

Por exemplo, para definir o tamanho de 12 "*pixels*" para a fonte dos textos das marcas *HTML* <p>, pode-se criar a seguinte regra *CSS*:

```
p { font-size: 12px; }
```

A Figura 5.3 mostra a visualização de uma página exemplo **sem** a aplicação da *CSS*. A Figura 5.4 mostra a mesma página **com** a aplicação da *CSS*.

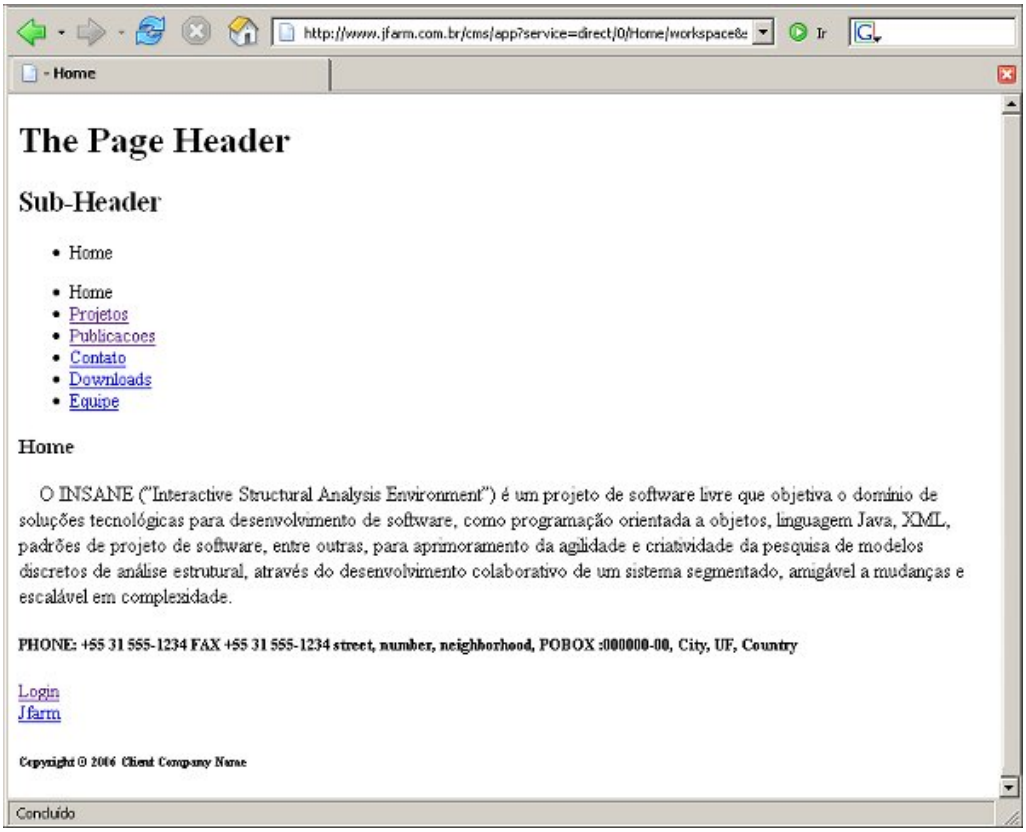


Figura 5.3: Página Web visualizada **sem** a aplicação das folhas de estilo (*CSS*).

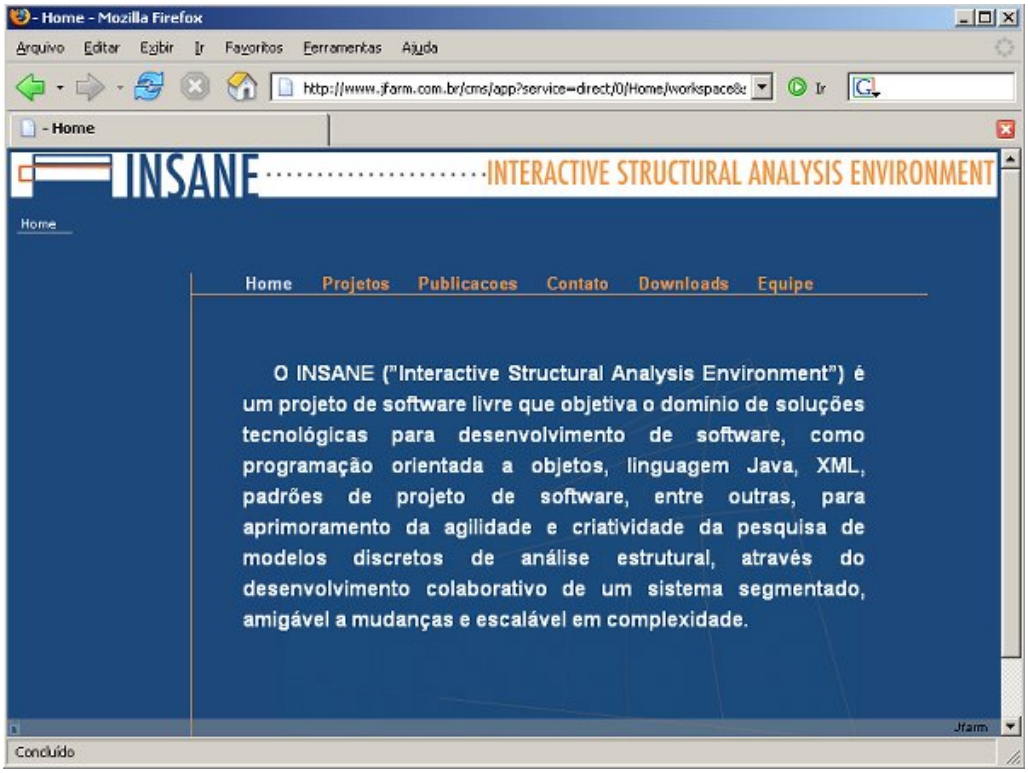


Figura 5.4: Página Web visualizada **com** a aplicação de uma folha de estilo (*CSS*).

Diferentes folhas de estilo fazem com que a mesma página Web fique com aparências completamente diferentes.

O uso das folhas de estilo em "*Sites* Web" traz algumas vantagens:

– Facilidade de manutenção do "*Site Web*", já que toda a formatação e estilo do *Site* pode ser definida em um único (ou em alguns) arquivos. Uma alteração de formato torna-se, assim, rápida e prática;

– Separação física e lógica do conteúdo da sua apresentação. Isso permite que pessoas diferentes fiquem responsáveis por cada parte do *Site* (conteúdo e apresentação);

– Possibilidade de personalização de um "*Web Site*" de acordo com usuários ou dispositivos (computadores, celulares, etc.), conforme mostra a Figura 5.5.

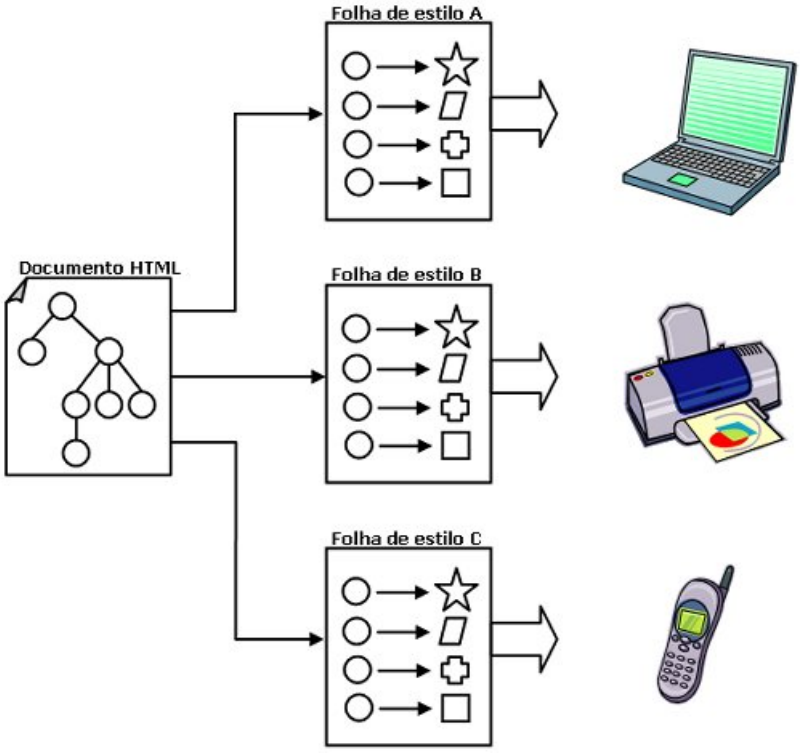


Figura 5.5: Combinação de folhas de estilo para diferentes finalidades (Ray, 2001).

Entretanto, alguns problemas podem aparecer quando as folhas de estilo são utilizadas. As regras da *CSS* estão sempre em evolução, propondo novos recursos e corrigindo problemas das versões anteriores. Os navegadores, que são os responsáveis por interpretar o código *HTML* e

as regras de estilo *CSS*, também estão sempre sendo atualizados e, às vezes, um determinado navegador (ou uma sua versão) ainda não implementa uma determinada especificação. O resultado é que um determinado recurso *CSS* pode funcionar em um navegador e não em um outro. Ele pode, até mesmo ter um resultado diferente em cada navegador. A tendência é que as versões mais novas dos navegadores implementem melhor as últimas especificações *CSS*.

As regras *CSS* podem ser definidas em um arquivo externo (com extensão .css) ou declaradas dentro das páginas (nos arquivos .html) usando importação ou mesmo escrevendo-as diretamente. Para que não haja dúvida sobre qual estilo será efetivamente utilizado em uma página, existe uma hierarquia na definição da prioridade de aplicação das regras da *CSS* (por isso ela é chamada de estilo em *cascata*):

1. Estilo inserido em um determinado elemento *HTML*: ocorre quando uma determinada formatação é inserida diretamente dentro do código específico de um elemento *HTML* (marca) em uma página (mais alta prioridade). No exemplo a seguir, este estilo é aplicado em um determinado parágrafo de modo a alterar a cor do texto para preto e alterar a margem para 5 "*pixels*":

```
<html>
<head>
    ........
</head>
<body>
    <p style="color:#000000; margin: 5px;">
    O INSANE ("Interactive Structural Analysis Environment"
    é ....
    </p>
    ........
</body>
</html>
```

2. Estilo incorporado em uma determinada página: ocorre quando um bloco de definições de estilo é disposto dentro da seção "*head*" do documento. Esta forma é ideal quando se quer alterar a aparência de um determinado documento apenas. No exemplo abaixo, a figura de fundo e as cores da marca <h3> e do fundo são definidas diferentemente no arquivo *HTML*:

```
<html>
<head>
    ........
    <style type="text/css">
        body { background-image: url("imagens/minhaimagem.gif"); }
        h3 {color: #FF0000; }
    </style>
    ........
</head>
```

3. Estilo externo: é aquele definido em um arquivo separado (de extensão .css). No exemplo abaixo, o estilo externo "`style1.css`" é associado a uma página:

```
<html>
<head>
    ........
   <link rel="stylesheet" type="text/css" href="insane/css/style1.css"/>
    ........
</head>
```

4. Estilo do usuário: definido quando o usuário altera as opções de exibição do seu navegador (por exemplo, aumentando o tamanho da fonte).

5. Estilo padrão do navegador do usuário (mais baixa prioridade).

As folhas de estilo podem ser combinadas em cascata para atender a necessidades especiais. Por exemplo, uma folha de estilo geral pode ser combinada com uma outra que ajusta as configurações de estilo para um produto específico (dispositivo sem fio, impressora, suporte para Braille, áudio, entre outros.), como mostra a Figura 5.6 (Ray, 2001).

As folhas de estilo também podem ser usadas para formatar um documento *XML*. Para associar uma folha de estilo a um documento *XML*, basta incluir a seguinte declaração no seu prólogo, logo após a declaração *XML*:

```
<?xml-stylesheet type="text/css" href="nomeFolha.css"?>
```

O valor do atributo "*href*" deve ser o nome do arquivo de folha de estilo a ser aplicado ao documento.

Todavia, existe um recurso muito mais poderoso que as *CSS* para formatar documentos *XML*, a *XSL* (*Extensible Stylesheet Language Family*), apresentada na seção 5.6.

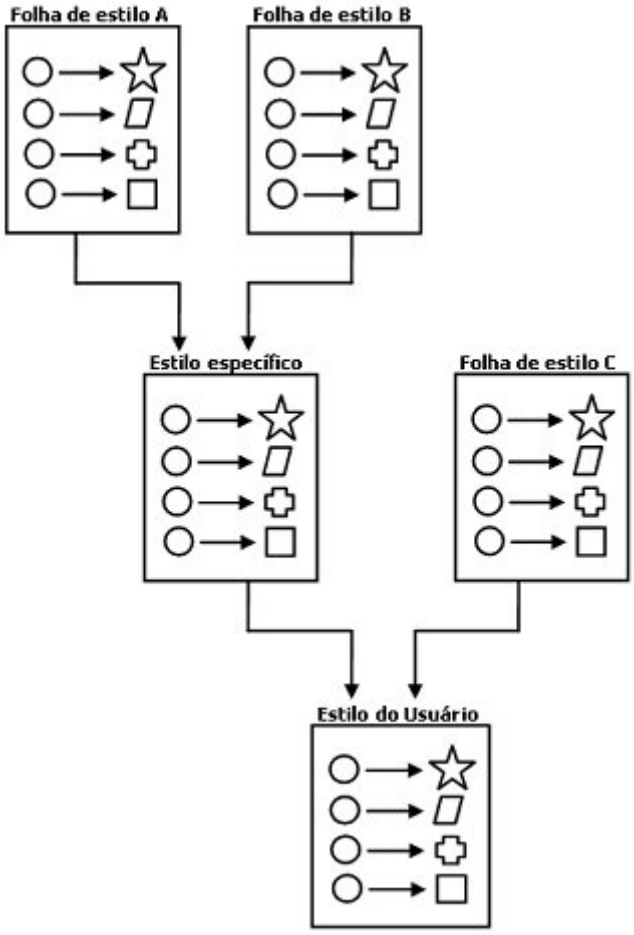


Figura 5.6: Uma cascata de folhas de estilo (Ray, 2001).

5.4 Tapestry

O *Tapestry* é uma poderosa biblioteca Java que permite criar aplicações Web dinâmicas, robustas e escaláveis (Tapestry, 2006).

O *Tapestry* permite uma programação de aplicativos Web de alto nível, pois encapsula toda a parte de mais baixo nível que trata dos protocolos de comunicação em rede.

Um aplicativo Web é formado por um conjunto de páginas *HTML* construídas com o uso de componentes. O componente é responsável por gerar dinamicamente conteúdo de partes da página Web, com o uso de classes Java. O *Tapestry* oferece por volta de 50 componentes, que são suficientes para atender às necessidades de um desenvolvedor Web (Smart, 2006). É possível, ainda, desenvolver novos componentes *Tapestry* para serem utilizados nas aplicações Web.

Uma página Web com conteúdo gerado dinamicamente pelo *Tapestry*, cujo nome seja **PaginaExemplo**, é formada a partir de três recursos:

- Modelo *HTML*: um arquivo com código *HTML* de nome `PaginaExemplo.html`. Neste modelo são inseridos os componentes *Tapestry*.

- Arquivo com as especificações da página e seus componentes: um arquivo escrito em *XML* de nome `PaginaExemplo.page`. Este arquivo é o responsável por fazer a ligação entre os componentes e as classes Java respectivas.

- Classes Java: arquivos de classes Java, de extensão ".class", que são os responsáveis por gerar o conteúdo dinamicamente. A classe principal será a de nome `PaginaExemplo.class`, mas poderá haver outras classes se for necessário à aplicação.

Em tempo de execução, o *Tapestry* lê o modelo *HTML* e identifica os componentes nele inseridos. Cada componente é processado e substituído por um valor que é obtido a partir de sua classe Java respectiva. Um arquivo *HTML* final é gerado com o conteúdo típico de um arquivo *HTML*, conforme mostrado na Figura 5.7, que é então enviado ao navegador e apresentado ao usuário.

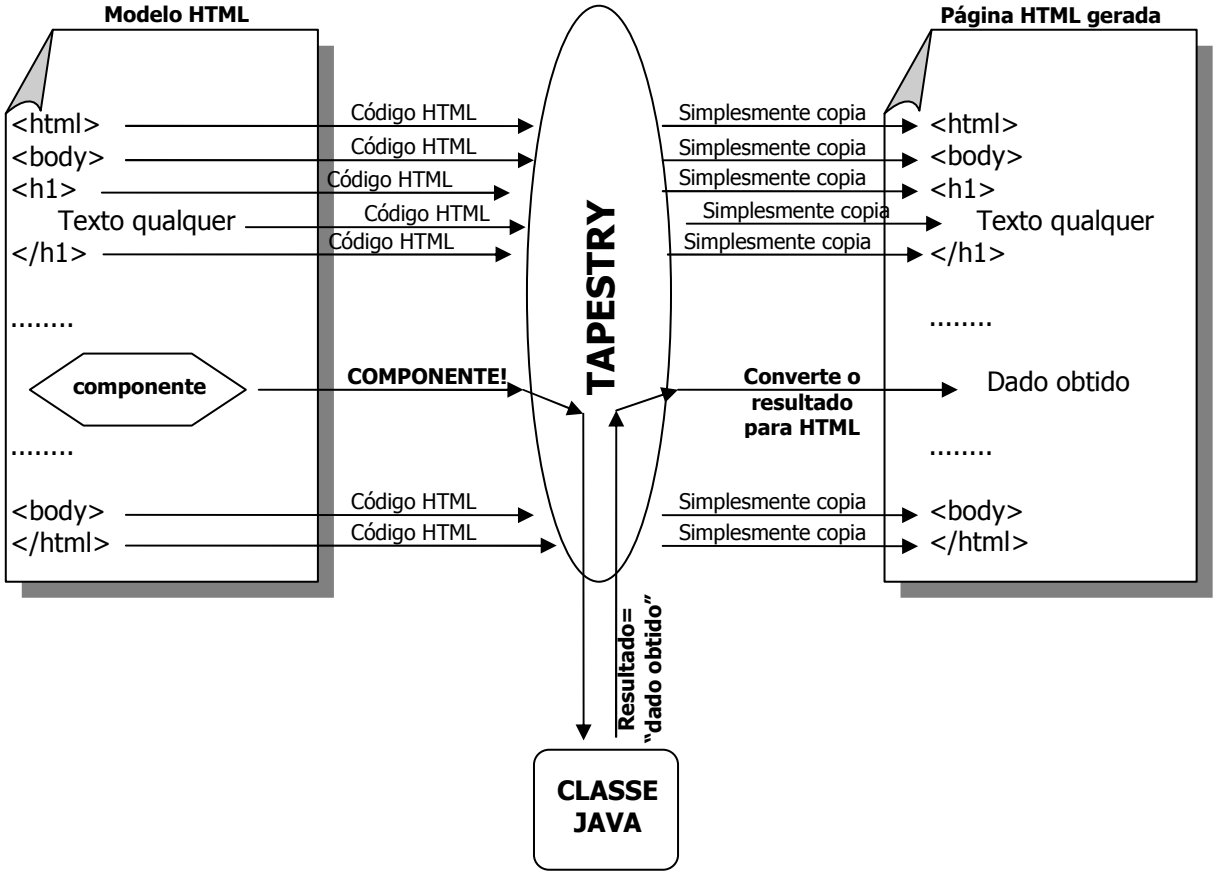


Figura 5.7: Geração dinâmica de páginas *HTML* com o uso do *Tapestry*.

5.4.1 Componente Tapestry

O componente *Tapestry* é identificado no código *HTML* da página modelo, pela palavra ***jwcid*** que significa: *Java Web Component ID*. Esta palavra é introduzida no código como um atributo de qualquer marca (*tag*) *HTML*. Quando não há uma marca na região onde se queira declarar um componente, usa-se a marca *HTML* ****. Esta marca não produz nenhum efeito ao seu conteúdo quando a página é visualizada pelo navegador, servindo apenas para introduzir um componente.

Os componentes *Tapestry* podem ser do tipo implícitos ou declarados.

– **Componentes implícitos:** são aqueles que descrevem o seu tipo e seus parâmetros diretamente no código *HTML*. No exemplo a seguir, é definido um componente (sem identificação) do tipo de inserção (*Insert*) cujo parâmetro "*value*" terá como valor o conteúdo

recuperado do atributo "*price*" da classe Java "`feature.Destination`". Este valor substituirá o número 199 pois o mesmo está envolvido pela marca <*span*>. O prefixo *ognl* indica que o que vem após os dois pontos é uma expressão *OGNL* (*Object Graph Navigation Language*), a qual informa ao ***Tapestry*** que o valor "`featureDestination.price`" é uma expressão a ser avaliada e não um texto comum.

```
<span jwcid="@Insert" value="ognl:featureDestination.price">199</span>
```

– **Componentes declarados:** são aqueles que têm o seu tipo e parâmetros definidos externamente, em um arquivo de especificações. Este arquivo tem o mesmo nome do modelo ***HTML*** seguido pela extensão ".page". Ele é um arquivo ***XML*** com a seguinte forma:

```
<?xml version="1.0"?>
<!DOCTYPE page-specification PUBLIC
    "-//Apache Software Foundation//Tapestry Specification 4.0//EN"
    "http://jakarta.apache.org/tapestry/dtd/Tapestry_4_0.dtd">
<page-specification class="contexto.Home">

</page-specification>
```

As linhas iniciais do arquivo ***XML*** formam o prólogo e identificam o documento como sendo um modelo de documento público ***Tapestry***. O bloco seguinte, formado pelo elemento raiz <page-specification> é onde devem ser inseridas as especificações dos componentes declarados. O atributo "*class*" identifica a classe Java principal. Seu valor deve ser o contexto (caminho de diretórios onde estão armazenadas as classes Java da aplicação) seguido por um ponto e pelo nome da classe, que tem o mesmo nome do arquivo ***HTML*** e do arquivo de especificações (neste exemplo a classe usada foi `Home.class`).

A marca a seguir, parte de um código ***HTML***, mostra um exemplo de componente declarado:

```
<html> Hello <span jwcid="subject">world</span>! </html>
```

O identificador do componente é "*subject*" e a sua especificação no arquivo de extensão ".page" está mostrada a seguir:

```
<page-specification class="com.ttdev.helloworld.Home">
    <component id="subject" type="Insert">
        <binding name="value" value="ognl:greetingSubject"/>
    </component>
</page-specification>
```

Dentro do bloco principal (elemento <page-specification>) há um elemento <component> que especifica o componente inserido no código *HTML*. Cada componente do tipo *declarado* será especificado através de um elemento com este nome. O atributo "*id*" tem como valor o identificador "*subject*" usado no código *HTML* e o atributo "*type*" tem como valor o tipo deste componente, que neste caso é de inserção (*Insert*). No corpo deste elemento, podem ser especificados um ou mais elementos <binding>. Um elemento <binding> é usado para definir um parâmetro associado ao tipo de componente. No exemplo, o parâmetro de nome "*value*" terá como valor o resultado da avaliação da expressão "ognl:greetingSubject". Para obter este resultado, o *Tapestry* acionará o método *getGreetingSubject()* da classe Java correspondente, neste caso a classe principal, `Home.class`, definida no atributo "*class*" do elemento <page-specification>.

Para a página deste exemplo, de nome "`Home.html`" e arquivo de especificações "`Home.page`", a classe Java correspondente (fonte) deve ter o nome "`Home.java`". Esta classe deve estender a classe "*BasePage*" do *Tapestry* para herdar uma série de funcionalidades desta plataforma. O Código 5.2 mostra a classe "`Home.java`" neste exemplo (Tong, 2005).

```
package com.ttdev.helloworld;

import org.apache.tapestry.html.BasePage;

public class Home extends BasePage {

    public String getGreetingSubject() {
        return "John";
    }
}
```

Código 5.2: Código fonte da classe `Home.java` (Tong, 2005).

Nesta classe é definido o método *getGreetingSubject()* que retornará um texto, neste caso uma "*String*" de valor "*John*", que será efetivamente inserida no código ***HTML*** final.

O Apêndice H, seção H.3 apresenta uma lista com alguns exemplos de componentes ***Tapestry*** além de outros recursos desta tecnologia.

5.4.2 Exemplo do funcionamento do Tapestry

A Figura 5.8 ilustra o funcionamento do ***Tapestry***, através da geração de uma página Web que exibe os resultados de um processamento feito por uma aplicação Web. Esta aplicação utiliza o ***Tapestry*** para calcular o vão de uma viga bi-apoiada a partir das coordenadas dos nós fornecidas pelo usuário.

O modelo desta página é o arquivo "`Result.html`". O componente "vao" que está ressaltado na figura (1), indica que um cálculo será feito pela aplicação e o valor do resultado obtido substituirá este componente na página a ser apresentada ao usuário. O arquivo de especificações "`Result.page`" (2) informa que o componente cujo identificador é "vao" será substituído pelo valor do atributo "vao" definido na classe Java "`Result.java`" (3). O ***Tapestry*** automaticamente acionará o método *getVao()* desta classe, para recuperar este valor, que será então introduzido na página ***HTML*** final e exibida ao usuário (4). Esta página está mostrada na Figura 5.9.

Na Figura 5.8 há ainda outro componente declarado implicitamente (5). Este componente é do tipo "*Insert*" e seu valor será recuperado a partir de outro método existente na classe "`Result.java`" cuja assinatura é: *getNomeEst()*. Na Figura 5.9 o resultado do processamento deste componente pode ser visualizado. O valor obtido foi o texto "viga 1".

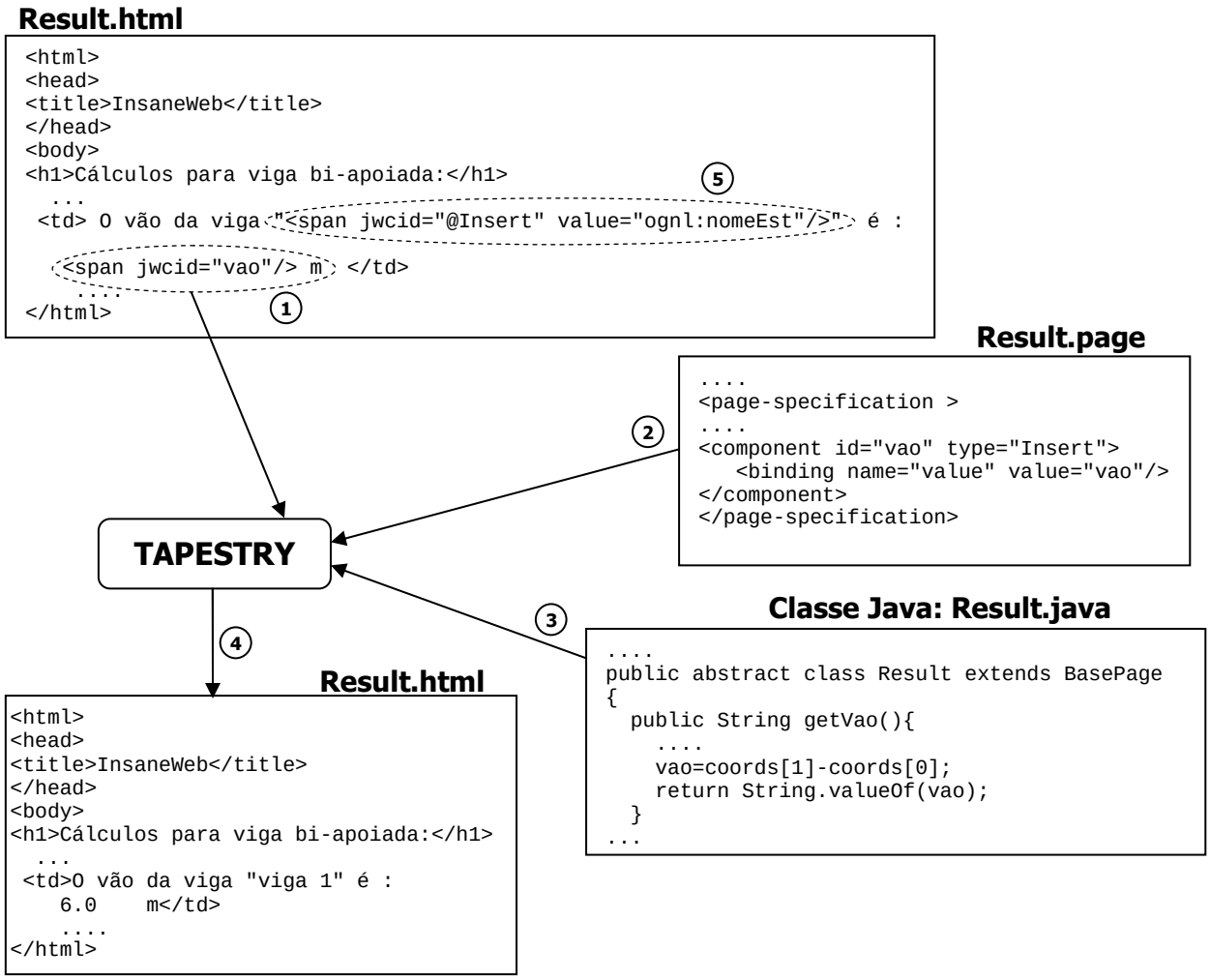


Figura 5.8: Esquema da geração dinâmica do código *HTML* com o uso do *Tapestry*.

Figura 5.9: Visualização de uma página Web gerada com o uso do *Tapestry*.

5.5 Tomcat

O Apache *Tomcat* é um servidor de aplicações Java para Web. É um *software* localizado no servidor que permite a execução de aplicações Web. É distribuído como *software* livre e desenvolvido como código aberto dentro do conceituado projeto Apache Jakarta e oficialmente endossado pela Sun como a Implementação de Referência (RI) para as tecnologias Java *Servlet* e *JavaServer Pages* (Tomcat, 2007).

Tecnicamente, o *Tomcat* é um servidor capaz de suportar o desenvolvimento e a execução, em ambiente de produção, de aplicações Web criadas segundo os padrões da plataforma Java (Lozano, 2004).

Uma aplicação Web é a denominação dada para um conjunto de elementos que são tipicamente condensados em um arquivo WAR (*Web ARchive*) para ser instalada em um contêiner Web. Estes elementos podem ser:

- `Servlets:` classes Java escritas para serem executadas em um servidor;
- **Páginas JSP** (*Java Server Pages*)**:** JSP é uma tecnologia para geração de conteúdo dinâmico em páginas Web através da inserção de código Java diretamente no arquivo *HTML*;
- **Arquivos estáticos:** figuras, documentos *HTML*, entre outros;
- **Classes auxiliares**;
- **Arquivo de configuração** (obrigatório)**:** é o descritor da aplicação que consiste em um arquivo *XML* com o nome "`web.xml`". Este arquivo especifica parâmetros de inicialização, regras de segurança e mapeamento de *URL*s para *Servlets*, entre outras informações. O Código 5.3 apresenta um exemplo deste arquivo de configuração, para uma aplicação Web chamada "*StockQuote*".

O *Tomcat* é um sistema complexo constituído por vários componentes, dentre os quais o mais importante é o *Catalina*, seu "Contêiner Web".

```
<?xml version="1.0"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/TR/xmlschema-1/"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  <display-name>StockQuote</display-name>
  <servlet>
    <servlet-name>StockQuote</servlet-name>
    <servlet-class>org.apache.tapestry.ApplicationServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>StockQuote</servlet-name>
    <url-pattern>/app</url-pattern>
  </servlet-mapping>
</web-app>
```

Código 5.3: Arquivo de configuração `web.xml` da aplicação Web *StockQuote* (Tong, 2005).

Um contêiner Web é um módulo que processa as requisições *HTTP* para um *Servlet* e monta as respostas *HTTP* para os aplicativos Web. Ele é responsável pela criação e destruição dos *Servlets* e pela delegação de requisições *HTTP* para os *Servlets* existentes.

A Figura 5.10 ilustra o fluxo de uma requisição *HTTP* com a invocação de um *Servlet*.

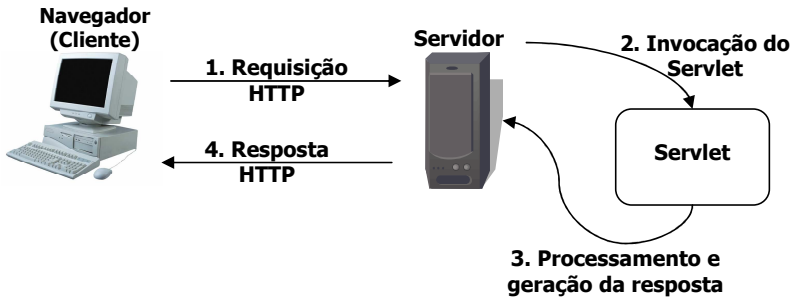


Figura 5.10: Fluxo de uma requisição *HTTP* com a invocação de um *Servlet*.

Tal fluxo ocorre em quatro etapas, enumeradas abaixo:

(1) A aplicação cliente (o navegador) envia uma requisição *HTTP* ao servidor Web;

(2) O servidor (seu módulo contêiner) invoca o *Servlet* apropriado;

(3) O *Servlet* recebe e processa a requisição e gera a resposta;

(4) O servidor envia a resposta à aplicação cliente.

Basicamente, o contêiner realiza três etapas para atender a uma requisição de um *Servlet* (Kurniawan e Deck, 2004):

- Cria um objeto de requisição com as informações que poderão ser usadas pelo *Servlet*, como parâmetros, cabeçalhos, cookies, *URI*, entre outras;
- Cria um objeto de resposta (vazio) que será usado pelo *Servlet* invocado para enviar o resultado de volta ao cliente Web;
- Invoca o método de serviço do *Servlet*, passando como parâmetros os objetos de requisição e resposta. O *Servlet* lê os dados do objeto de requisição e grava os resultados no objeto de resposta.

O *Tomcat* é inicializado na máquina do servidor e fica esperando uma chamada de um cliente Web (uma requisição *HTTP* para alguma aplicação Web). Ele possui uma aplicação de administração (*Tomcat Manager*) que gerencia as várias aplicações Web hospedadas no servidor (ver Figura 5.11).

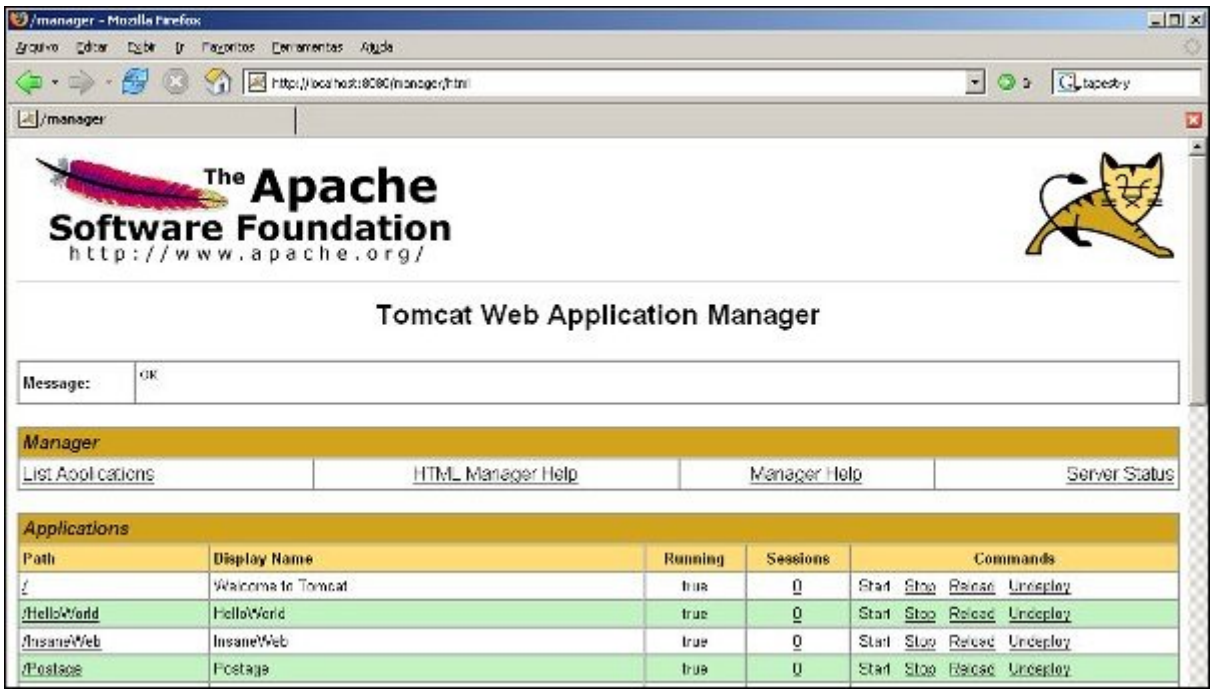


Figura 5.11: Exemplo da aplicação de administração do *Tomcat* (*Tomcat Manager*)

5.6 XSL

A *XSL* (*Extensible Stylesheet Language*) ou Linguagem de Folhas de Estilo Extensível é muito mais que uma linguagem de formatação. Ela é uma família de recomendações para transformação e apresentação de documentos *XML*. Ela é uma linguagem escrita em *XML* que oferece uma série de recursos para manipulação de dados, como filtragem e ordenação e pode oferecer o resultado final em vários formatos, como *HTML*, *XML*, PDF, entre outros.

A Figura 5.12 ilustra o funcionamento da transformação de documentos *XML*. Um documento *XML* é transformado por um processador *XSLT* a partir de um documento *XSL*. A associação a este documento pode ser feita através do próprio documento ou pelo processador. O arquivo resultante pode ser um outro documento *XML* ou um documento *HTML* ou ainda, outros formatos, como PDF, formatos tipo texto, entre outros.

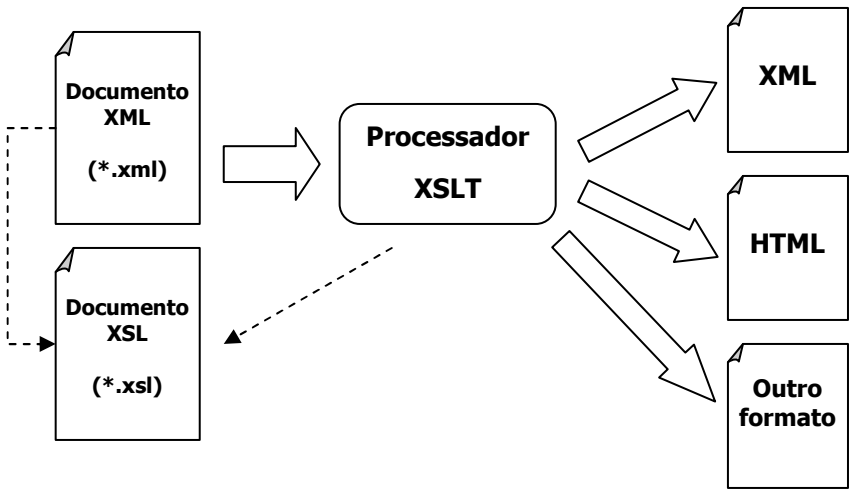


Figura 5.12: Transformação de documentos *XML* com a *XSLT*.

A associação ao *XSL* pode ser feita dentro do documento *XML* através da inclusão de uma linha com a referência ao arquivo *XSL*, como mostrado na segunda linha do trecho de código a seguir:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
    ....
</catalog>
```

Deste modo, quando o arquivo *XML* for visualizado por um programa que suporte tanto *XML* quanto *XSL*, a transformação ocorre automaticamente. Alguns navegadores Web são compatíveis com estas duas linguagens, como o *Internet Explorer* (versões 6 em diante), *Firefox*, entre outros.

O arquivo *XML* de exemplo "`cdcatalog.xml`" mostrado no Código 5.4, contém alguns dados em formato *XML*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
.
</catalog>
```

Código 5.4: Arquivo `cdcatalog.xml`.

O arquivo *XSL* de exemplo "`cdcatalog.xsl`" mostrado no Código 5.5, contém as regras de transformação para o arquivo *XML* do Código 5.4.

A Figura 5.13 mostra o documento *XML* do Código 5.4 quando é visualizado pelo navegador *Firefox*.

Neste caso, o documento *XML* foi convertido em uma página *HTML*. Os exemplos foram retirados de w3Schools (2007*b*).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
    <tr bgcolor="#9acd32">
      <th align="left">Title</th>
      <th align="left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Código 5.5: Arquivo `cdcatalog.xsl`.

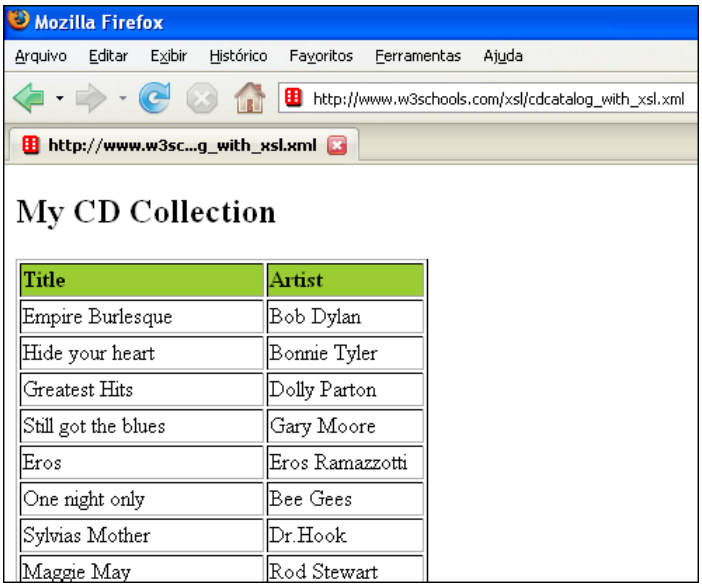


Figura 5.13: Visualização do documento *XML* com a *XSLT*.

A *XSL* se baseia em três componentes, segundo as recomendações da *W3C* (`http://www.w3.org/Style/XSL/`), os quais estão listados resumidamente a seguir e detalhados no Apêndice H, seção H.4.

5.6.1 XPath

A *XPath* ou "*XML Path Language*" é uma linguagem para acessar e referenciar elementos e atributos em um documento *XML*.

Os elementos *XML* são referenciados no *XPath* através de expressões de caminho, parecidas com as expressões de sistemas de arquivos.

A expressão *XPath* a seguir seleciona todos os elementos *price* de todos os elementos *cd* do elemento *catalog* do documento *XML* mostrado no Código 5.4.

```
/catalog/cd/price
```

A *XPath* também define uma biblioteca de funções para manipulação de textos (cadeias de caracteres) e números, além de expressões booleanas. O exemplo a seguir, seleciona todos os elementos *cd* que têm um elemento *price* cujo valor seja maior que 10.80.

```
/catalog/cd[price>10.80]
```

5.6.2 XSLT

A *XSLT* ou "*XSL Transformations*" é uma linguagem para transformações de dados *XML*. Com esta linguagem, é possível transformar documentos *XML* em outros tipos de documentos, como documentos que são reconhecidos pelos navegadores (*HTML* ou *XML*) ou documentos em formato texto comum, PDF, entre outros. Durante a transformação, a *XSLT* permite filtrar dados, reordená-los, testar, adicionar novas informações, além de outras possibilidades.

As regras *XSLT* de transformação são definidas em um arquivo *XSL*, que é um arquivo texto escrito em *XML*, no qual expressões *XPath* são usadas para montar os padrões de transformação. Neste arquivo, as regras são montadas em partes chamadas de moldes. Durante a transformação, quando os dados *XML* encontram uma combinação com os moldes, o resultado é produzido.

Um arquivo *XML* é identificado como *XSL* quando o elemento raiz é a palavra "*stylesheet*" ou "*transform*", além de ter declarado seu *namespace* de acordo com as recomendações W3C, como mostrado no trecho de código abaixo:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

No restante do documento *XSL* são declarados os moldes, que contém as regras a serem aplicadas ao documento *XML*. Um molde é definido usando-se o elemento <xsl:template>. No Código 5.5 o elemento template será aplicado a todos os elementos *XML*, pois a expressão *XPath* usada foi:

```
<xsl:template match="/">
```

O símbolo da barra sozinho indica que o molde será válido em todos os elementos *XML*. Observa-se, neste exemplo, os elementos *HTML* adicionados (<html>,<body>,<h2>, entre outros) e as primeiras regras de filtragem definidas:

```
<td><xsl:value-of select="catalog/cd/title"/></td>
<td><xsl:value-of select="catalog/cd/artist"/></td>
```

Nesta duas linhas, o elemento <xsl:value-of> extrai os valores dos elementos *title* e *artist* através das expressões *XPath* mostradas acima. Para que todos os elementos *cd* sejam avaliados, o elemento de laço <xsl:for-each> é utilizado e uma expressão *XPath* limita esta avaliação aos elmentos *cd*:

```
<xsl:for-each select="catalog/cd">
```

A transformação dos documentos *XML* ocorre quando um processador *XSLT* é executado. O navegador pode funcionar como um processador quando há uma associação a um documento *XSL* dentro do arquivo *XML*. Entretanto, o navegador precisa ser compatível com a especificação *XSLT*. Como alternativa, existem programas processadores *XSLT* disponíveis (alguns dos quais são gratuitos), como os listados a seguir:

(i) Xalan: versões para Java (`http://xml.apache.org/xalan-j/`) e C++ (`http://xml.apache.org/xalan-c/`);

(ii) Saxon: `http://www.saxonica.com/`;

(iii) Oracle: `http://www.oracle.com/technology/tech/xml/index.html`;

(iv) XT: `http://www.blnz.com/xt/index.html`.

É possível, ainda, realizar as transformações internamente em programas através de *APIs* que fazem o papel de processadores. Para a linguagem Java, há um conjunto de *APIs* que realizam esta função, as quais estão listadas a seguir:

(i) `javax.xml.transform`;

(ii) `javax.xml.transform.stream`;

(iii) `javax.xml.transform.dom`;

(iv) `javax.xml.transform.sax`.

Deste modo, basta criar um código que leia o arquivo *XML*, o arquivo *XSL* e realize a transformação *XSLT*. O Código 5.6 mostra o trecho de uma classe Java que faz esta transformação.

Em aplicações Web, é possível realizar a transformação *XSLT* através do uso de *JavaScripts*, permitindo, assim, que ela ocorra no lado do cliente. Porém, essa opção só funcionará em navegadores que suportem as *APIs* utilizadas. Realizando a transformação no lado do servidor, através de programação apropriada embutida na aplicação Web, o resultado pode ser uma página *HTML* que será visualizada por qualquer navegador.

```
public static String transformXml(String xmlFileName, InputStream xslFileIS){
  XMLInputFactory inputFactory=XMLInputFactory.newInstance();
  XMLOutputFactory outputFactory=XMLOutputFactory.newInstance();
  try{
    XMLStreamReader streamReaderXSL=
      inputFactory.createXMLStreamReader(xslFileIS);
    TransformerFactory transfFactory=TransformerFactory.newInstance();
    Source XSL=new StAXSource(streamReaderXSL);
    Transformer transf=transfFactory.newTransformer(XSL);
    transf.setOutputProperty(OutputKeys.INDENT, "yes");
    Source XML=new StreamSource(new File(xmlFileName));
    XMLStreamWriter streamWriter=
      outputFactory.createXMLStreamWriter(new FileWriter(resultFileName));
    Result transResult=new StAXResult(streamWriter);
    transf.transform(XML,transResult);
        ...
    }catch(Exception e)
    {...}
}// end of transformXml
```

Código 5.6: Classe Java para transformação *XSLT* (`transform.java`).

5.6.3 XSL-FO

A *XSL-FO* ou "*XSL Formatting Objects*" é uma linguagem para formatação de dados *XML* para exibição em telas, papéis e outras mídias. Com ela é possível definir o *layout* de página, fontes, estilos, cores, imagens e muitas outras propriedades.

Um documento *XSL-FO* é composto por:

(i) Um cabeçalho *XML* e uma declaração de *namespace* apropriada: `<fo:rootxmlns:fo= "http://www.w3.org/1999/XSL/Format">`;

(ii) Informações de *layout* de página;

(iii) Definições de cabeçalhos e rodapés;

(iv) Conteúdo (texto).

A Figura 5.14 mostra o esquema do *layout* de página do *XSL-FO* (w3Schools, 2007*a*).

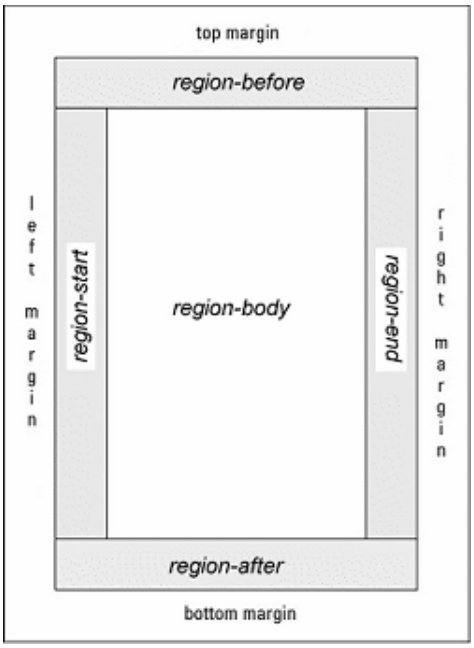


Figura 5.14: Esquema do *layout* de página do *XSL-FO* (w3Schools, 2007a).

Uma página, conforme ilustrado na Figura 5.14, é composta por cinco regiões: o corpo (*region-body*), anterior (*region-before*), posterior (*region-after*), início (*region-start*) e fim (*region-end*).

O Código 5.7 mostra um exemplo arquivo *XSL-FO*.

O elemento <fo:layout-master-set> define a seção onde são definidos os *layouts* de página, que, por sua vez, são definidos pelo elemento <fo:simple-page-master>. Nesta seção, são definidos os tamanhos de folha e margens. A seguir, são definidos os elementos <fo:page-sequence> que descrevem o conteúdo efetivo das páginas. O atributo "master-reference" associa a esta página um dos *layouts* de página definidos na seção anterior. O elemento <fo:flow> define o conteúdo do documento de resultado. Ele pode conter vários elementos <fo:block>, os quais definem partes do texto, como parágrafos, tabelas, listas, entre outros.

Um documento *XSL-FO* pode ser usado para transformar dados *XML*, quando ele contém apenas instruções de formatação e expressões *XPath* e *XSLT* são usadas para definir os moldes. Assim, com o uso de um processador *XSLT*, o documento *XSL-FO* pode ser usado para obter-se um documento de saída formatado, que pode ser um arquivo PDF, PS, *HTML*, entre outros.

O uso do *XSL-FO* apresenta as seguintes vantagens:

```
<?xml version="1.0" encoding="utf-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
        <fo:simple-page-master master-name="LetterPage"
                    page-width="6in" page-height="5in">
            <fo:region-body region-name="PageBody" margin="0.7in"/>
        </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="LetterPage">
        <fo:flow flow-name="PageBody" font-family="Arial"
                        font-size="12pt" >
            <fo:block text-align="justify" space-after="0.5cm"
                      border="0.5pt solid green" >
                This is a text content.
            </fo:block>
            <fo:block text-align="justify" space-before="2cm"
                      border="0.5pt dotted red" >
                This is the other text content.
            </fo:block>
        </fo:flow>
    </fo:page-sequence>
</fo:root>
```

Código 5.7: Exemplo de arquivo *XSL-FO*.

(i) Divisão da página em várias colunas;

(ii) Geração de listas;

(iii) Controle de paginação, ou seja, permite definir que partes do texto podem ser separadas e que partes devem permanecer juntas;

(iv) Definição de cabeçalhos e rodapés para as páginas;

(v) Geração de tabelas;

(vi) Geração automática de índice a partir de marcas específicas.

(vii) Geração de bordas, cores de fundo, inserção de elementos gráficos, entre outros recursos.

Existem alguns processadores *XSL-FO* disponíveis, alguns dos quais estão listados a seguir:

(i) Apache FOP: oferece várias opções de saída além do PDF (`http://xmlgraphics.apache.org/fop/`);

(ii) Ecrion *XSL-FO* Engine: também oferece várias opções de saída (`http://www.ecrion.com`);

(iii) *XSL* Formatter: oferece a opção de saída em PDF (`http://www.antennahouse.com/`);

(iv) Xinc Beta Release: visualizador baseado em *Swing* e oferece o formato PDF como saída (`http://www.lunasil.com/`);

(v) Scriptura: `http://www.scriptura-xsl.com/`.

Capítulo 6

NÚCLEO NUMÉRICO DO INSANE COMO SERVIÇO WEB

6.1 Introdução

O projeto **INSANE** (`http://insane.dees.ufmg.br`) visa desenvolver um sistema computacional de análise de modelos discretos de elementos finitos. Utilizando-se modernos recursos tecnológicos, o sistema, a cada novo trabalho de um colaborador, amplia sua complexidade através da incorporação de uma nova ferramenta, modelo, interface ou solução desenvolvida no seu trabalho. É importante ressaltar que, a cada evolução, o sistema é expandido sem a necessidade de se refazer nenhuma implementação previamente desenvolvida.

O sistema **INSANE** é formado por aplicativos que podem ser classificados em três grandes segmentos: pré-processadores, processador e pós-processadores.

Os aplicativos são implementados na linguagem JAVA segundo o paradigma de programação orientada a objetos (POO), adotando-se a combinação mostrada na 6.1 de arquitetura em camadas e padrões de projeto de software (Modelo-Vista-Controlador (MVC), *Observer-Observable* e *Command*), conforme mostra a Figura 6.1. Maiores detalhes sobre esta combinação e os padrões utilizados podem ser encontrados em Penna (2007), Gonçalves (2004), Fonseca (2004) e Moreira (2004). Esta alternativa é bastante apropriada uma vez que permite a separação do processamento da informação de sua representação gráfica, facilitando assim os trabalhos de expansão e manutenção da aplicação.

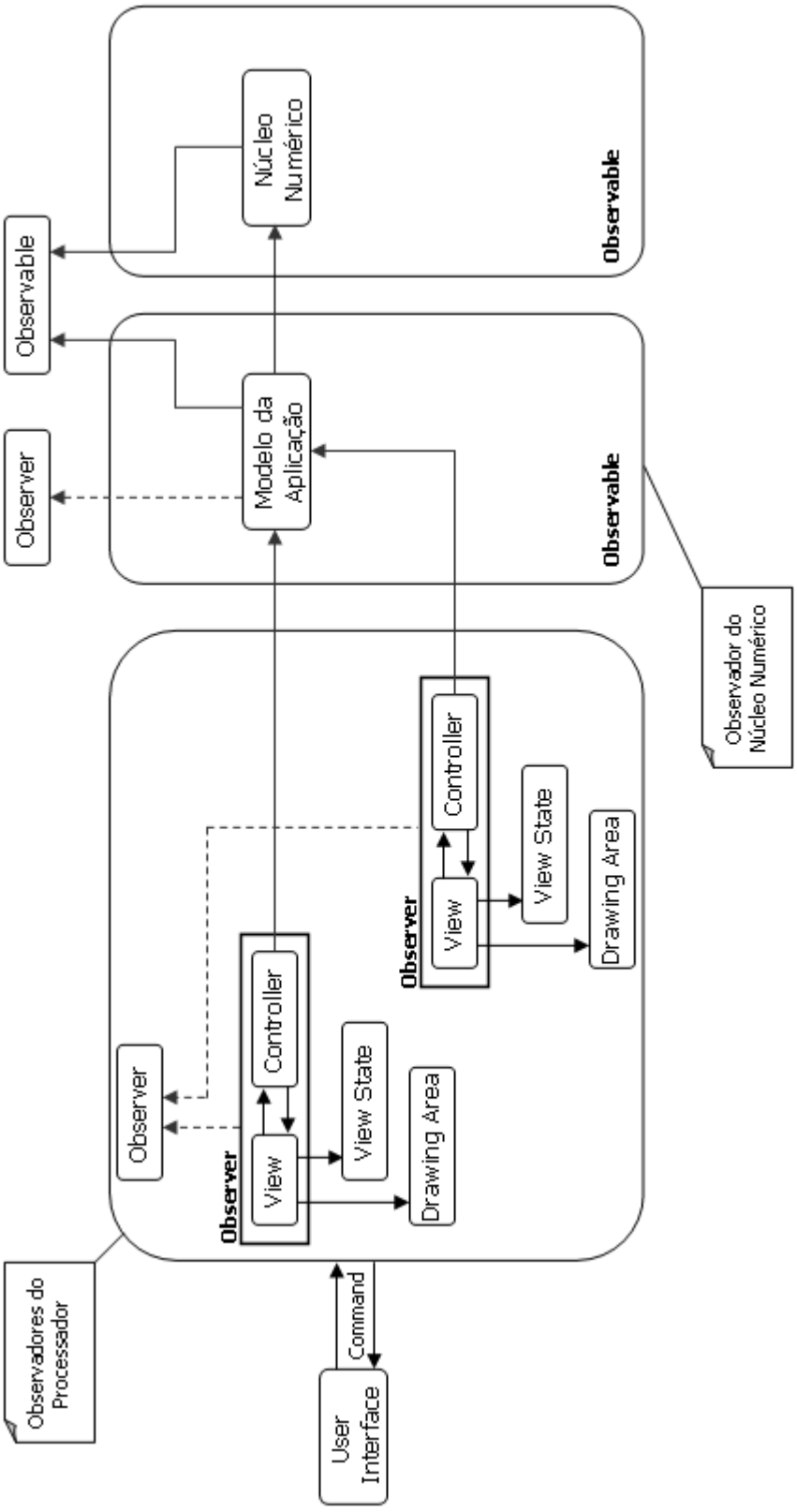


Figura 6.1: Arquitetura em camadas e padrões de projeto adotados no **INSANE**.

Em termos físicos, cada aplicativo **INSANE** possui atualmente somente duas camadas: uma aplicação carregada na memória do computador (compreendendo as camadas lógicas Modelo, Vista e Controlador) e arquivos textos e/ou binários persistidos em disco.

O presente trabalho disponibiliza o núcleo numérico do **INSANE** como um Serviço Web a ser consumido por aplicações diversas. O Capítulo 7 mostra alguns exemplos de clientes para este Serviço Web.

Neste novo formato, ilustrado na Figura 6.2, as camadas vista e controlador passam a ser de responsabilidade das aplicações consumidoras e a camada modelo (neste caso, o núcleo numérico) passa a ser o Serviço Web. Nestas aplicações, o modelo da aplicação (ver Figura 6.2) pode ou não existir. Assim, o sistema passa a ter três camadas físicas: a aplicação cliente (compreendendo as camadas lógicas Vista e Controlador), o Serviço Web (compreendendo a camada lógica do modelo) e arquivos textos e/ou binários persistidos em disco.

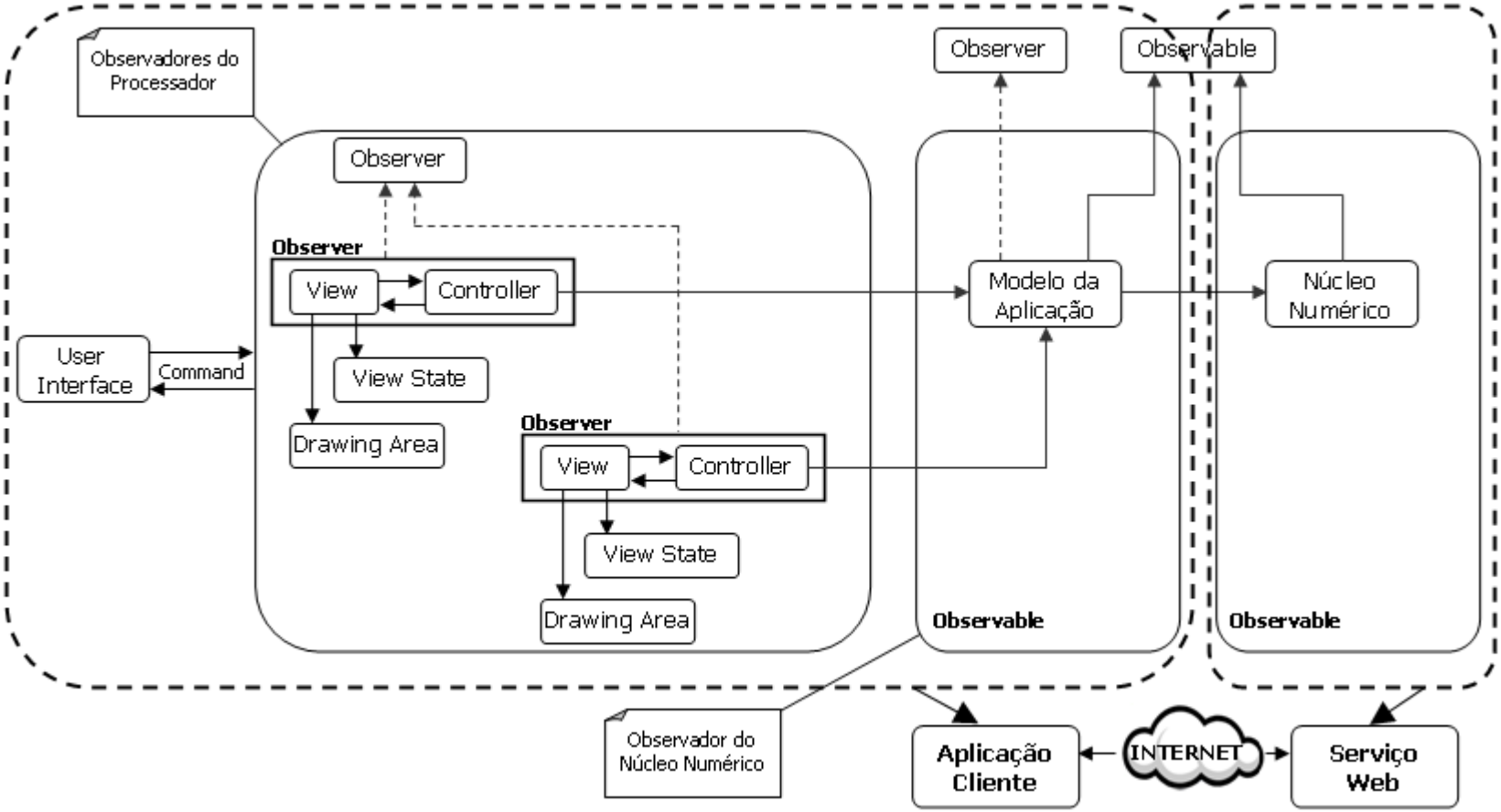


Figura 6.2: Nova arquitetura do projeto **INSANE**.

6.2 Generalização

A idéia principal deste trabalho, é disponibilizar o núcleo numérico do **INSANE** como um Serviço Web, funcionando como uma função (ou procedimento) a ser invocado por programas clientes, mantendo todas as implementações deste núcleo anteriormente feitas. Como o projeto **INSANE** segue o modelo de orientação a objetos e o núcleo numérico é implementado de maneira totalmente independente das aplicações de pré e pós-processamentos, a implementação deste novo projeto foi muito simples, necessitando apenas de uma classe Java, que faz o papel de interface entre o processamento remoto e as classes Java do núcleo numérico existentes. Isto é bastante apropriado ao projeto como um todo, visto que o mesmo está em constante desenvolvimento, e a atualização das classes do núcleo numérico torna-se bastante simples, como mostrado na seção 6.7.

Para adequar o sistema **INSANE** ao conceito de Serviço Web, a autora participou da fase inicial (meados de 2005) da generalização do modelo com o objetivo de tornar o **INSANE** um sistema com interfaces claras e genéricas. Outros desenvolvedores do grupo **INSANE** deram continuidade à esta generalização de acordo com as necessidades encontradas durante a elaboração de seus trabalhos, como as dissertações de mestrado de Fonseca (2006) e Saliba (2007) e o projeto de tese de doutorado de Fuina (2006).

Esta generalização contribui para que o sistema contemple diversos tipos de problemas, como, por exemplo, os problemas de dinâmica estrutural, os de campo generalizados e os de mecânica dos fluidos, além dos problemas de elasticidade de mecânica dos sólidos.

Na forma discreta do MEF, este problema, e tantos outros, podem ser descritos através do seguinte sistema de equações algébricas:

$$[A]\{\ddot{X}\} + [B]\{\dot{X}\} + [C]\{X\} = \{R\} \tag{6.1}$$

onde {X} é a variável de estado do problema e cada termo depende do significado do mesmo. O Apêndice A apresenta exemplos de problemas de computação científica que podem ser modelados com a generalização aqui apresentada.

6.3 Projeto OO do Núcleo Numérico

Apesar do projeto orientado a objetos do núcleo numérico não fazer parte do escopo deste trabalho, apresenta-se aqui uma breve descrição do mesmo.

O núcleo numérico do **INSANE** é hoje (agosto de 2007), composto por um conjunto de classes e interfaces, sendo que as principais estão representadas no diagrama de classes da Figura 6.3.

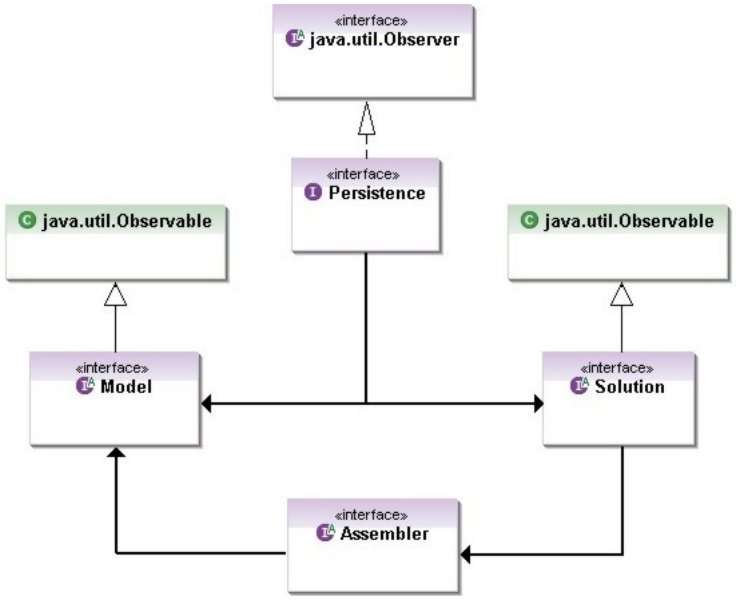


Figura 6.3: Organização do núcleo numérico do **INSANE** (Fonseca e Pitangueira, 2007).

A Interface ***Persistence*** padroniza a composição dos dados dos modelos através de arquivos textos (*XML*) ou binários (ISN) e a Interface ***Model*** representa os modelos a serem solucionados, possuindo os dados que compõem o problema. Além disso, a Interface ***Assembler*** monta o sistema de equações do problema buscando no modelo os dados necessários, e a Interface ***Solution*** define os métodos para solucionar a equação do problema.

Conforme ilustrado na Figura 6.3, tanto a Interface ***Model*** como a ***Solution*** se comunicam com a Interface ***Persistence***, que persiste os dados de saída sempre que observa alterações no estado do modelo discreto (Fonseca e Pitangueira, 2007). Este processo de observação de alterações ocorre segundo o padrão de projeto *Observer-Observable*, que é um mecanismo de propagação de mudanças. Quando um objeto dito observador (que implementa a interface

`java.util.observer`) é criado, ele é inscrito na lista de observadores dos objetos ditos observados (que estendem a Interface `java.util.observable`). Quando alguma mudança ocorre no estado de um objeto observado, é disparado o mecanismo de propagação de mudanças, que se encarrega de notificar os objetos observadores para se atualizarem. Isto garante a consistência e a comunicação entre o componente observador (***Persistence***) e os componentes observados (***Solution*** e ***Model***) (Fonseca, 2006).

O Apêndice B apresenta mais detalhes do núcleo numérico do **INSANE** mostrando suas principais classes e interfaces.

6.4 Codificação do Serviço Web

Conforme exposto no Capítulo 3, seção 3.4, o primeiro passo no desenvolvimento do Serviço Web **INSANE** é a definição da interface do Serviço e dos métodos a serem disponibilizados ao público (fase de codificação). São estes métodos que poderão ser acessados por outros programas para que estes usufruam do núcleo numérico **INSANE**.

Para o desenvolvimento deste Serviço Web, foi criado um novo projeto **INSANE**, desenvolvido em Java, denominado "`br.ufmg.dees.insane.webServiceWS`".

O primeiro método pensado para ser disponibilizado pelo Serviço Web foi o de resolução de modelos **INSANE**, ou seja, a partir de um arquivo de dados fornecido pela aplicação cliente, o Serviço Web **INSANE** pode ser utilizado para solucionar o modelo remotamente e retornar à aplicação cliente os resultados do processamento. Assim, foi implementado o método *getModelSolved*. O funcionamento desta opção do Serviço Web proposto está ilustrado na Figura 6.4.

A aplicação cliente lê o arquivo *XML* contendo os dados do modelo **INSANE** (1). Estes dados são inseridos em uma mensagem *SOAP* e enviadas ao Serviço Web através de uma requisição *HTTP* (2). O Serviço Web aciona o método solicitado, *getModelSolved*, que aciona as classes do projeto **INSANE** passando, como parâmetro, os dados *XML* enviados pelo cliente (3). O processamento é realizado no Servidor e os arquivos de resultados são gerados (4). O Serviço Web coleta estes arquivos e os prepara para enviar como resposta, convertendo-os em

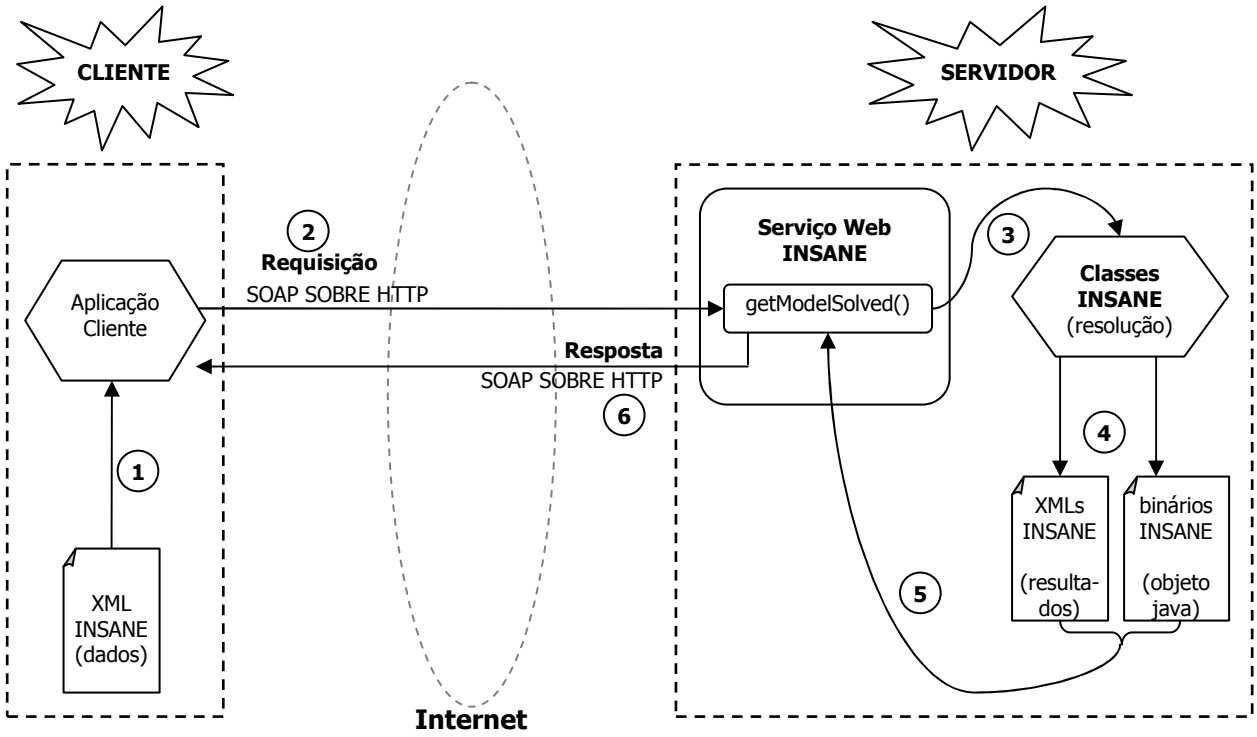


Figura 6.4: Esquema do funcionamento do Serviço Web **INSANE** - Método *getModelSolved*.

formato binário, ou seqüência de *bytes* (5). O Serviço monta, então, a mensagem *SOAP* de resposta com os arquivos gerados e a envia de volta ao cliente utilizando uma resposta *HTTP* (6).

Portanto, a interface Java a partir da qual o descritor *WSDL* será gerado, está apresentada no Código 6.1.

```
public interface InsaneService {

  /**  This method offers the possibility of the Insane Model resolution using the
   * Insane classes
   *
   * @param documentElement The Insane root XML element
   * @param modelName A model Name
   * @return   */
   public abstract OMElement getModelSolved(OMElement documentElement,String modelName);

   /** This method offers the possibility of getting a picture representation of the
    * Insane Model
    * @param fileElement A XML element with the .xml file  in binary mode!
    * @return a PNG figure of the model    */
    public abstract OMElement getFigure(OMElement fileElement);

    /** This method offers the possibility of getting a picture representation of the
     * Insane Model Results
     *
     * @param fileElement A XML element with the .isn file in binary mode!
     * @param label The String label of the Result wanted to be drawn
     *
     * @return a PNG figure of the model result     */
     public abstract OMElement getResultFigure(OMElement fileElement, String label);

     /** This method returns a list of the modelResult labels from a .isn file (solved).
      *
      * @param fileElement A XML element with the .isn file in binary mode!
      *
      * @return String[] with the model result labels      */
      public abstract String[] getModelKeys(OMElement fileElement);
}
```

Código 6.1: Interface `InsaneService.java`.

Nesta interface, observa-se o método *getModelSolved* acima mencionado, além de outros três métodos que foram necessários para aprimorar o cliente Web desenvolvido, o qual está mais detalhado no Capítulo 7.

O método *getModelSolved* recebe dois parâmetros:

1. parâmetro *documentElement* de tipo `OMElement` que representa um elemento *XML* genérico (AXIOM, 2007) contendo o elemento **INSANE** e todos os seus sub-elementos, conforme preconiza o *XML Schema* **INSANE** (Apêndice D);

2. parâmetro *modelName* do tipo `String` que representa um título alfanumérico para o modelo.

Além disso, o método *getModelSolved* retorna ao cliente, uma lista de arquivos de resultados (arquivos texto *XML* e arquivos binários) representada pelo elemento `OMElement` de retorno. É importante notar que, até esta fase, ainda não há nenhuma implementação da lógica do negócio (corpo do código do Serviço Web), pois a classe *Skeleton* ainda não foi gerada. Como em toda interface Java, apenas a assinatura do método foi definida, o que basta para o próximo passo de geração do descritor *WSDL*, o qual precisa conhecer e expor apenas os tipos de dados de entrada e saída.

Definidos os métodos a serem disponibilizados pelo Serviço Web, procede-se, então, à geração do descritor *WSDL* do mesmo. Neste trabalho, conforme exposto no Capítulo 4, seção 4.7, utilizou-se a plataforma *Axis*2, para a linguagem Java, devido às grandes facilidades oferecidas por ela e, para gerar o *WSDL*, o *plug-in* "*Axis*2 Eclipse Codegen" disponível em `http://ws.apache.org/axis2/tools/index.html`. A Figura 6.5 ilustra como esse processo é executado.

O *WSDL* gerado pode ser visto no Apêndice F e também pelo endereço Web `http://insane.dees.ufmg.br:8080/axis2/services/InsaneService?wsdl`.

Novamente, com o auxílio da ferramenta *Axis*2, a partir do arquivo *WSDL*, gera-se, então, as classes Java do Serviço Web (*Skeleton* e classes auxiliares) e o seu arquivo de configuração "`services.xml`" (conforme mencionado na seção 4.7.1). Este procedimento está apresentado na Figura 6.6.

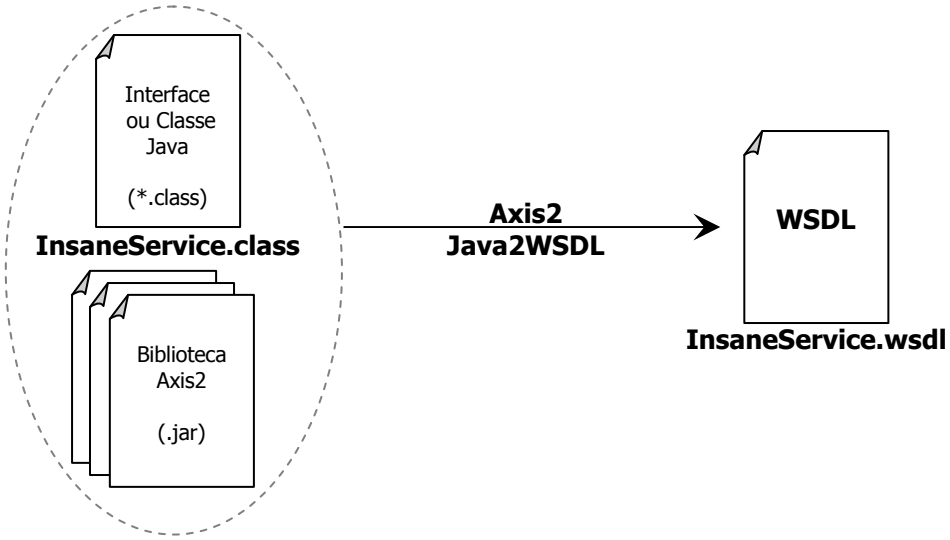


Figura 6.5: Geração do *WSDL* do Serviço Web **INSANE**.

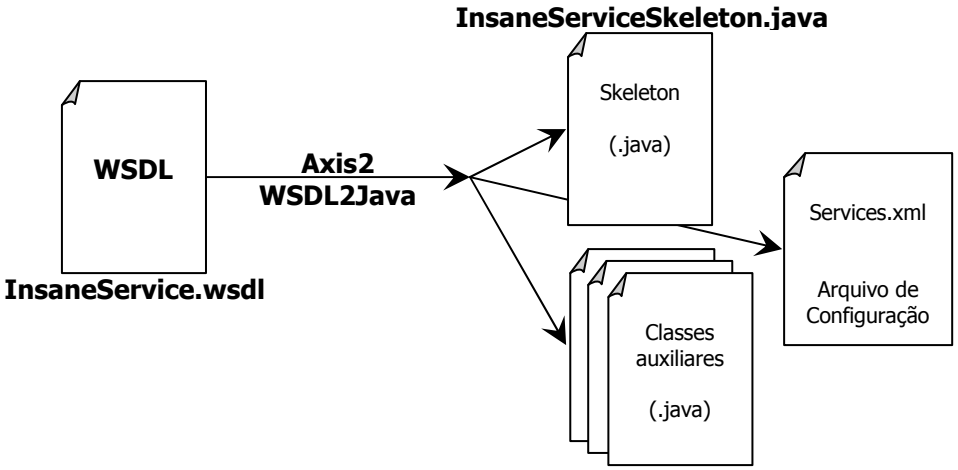


Figura 6.6: Geração das classes Java do Serviço Web **INSANE** (*Skeleton*).

O código da classe *Skeleton* pode então ser acrescentado.

A parte da classe *Skeleton* correspondente ao método *getModelSolved* que é o responsável pelo principal objetivo do Serviço Web (resolução de modelos **INSANE**), realiza as seguintes tarefas, ilustradas na Figura 6.7:

1. Recebe o arquivo *XML* enviado pelo cliente e o salva no servidor;

2. Aciona a classe ***Persistence*** do núcleo numérico **INSANE**, passando a referência ao arquivo *XML* mencionado acima;

3. O modelo é, então, resolvido (3a) e os arquivos de resultados são gerados (3b) ou uma

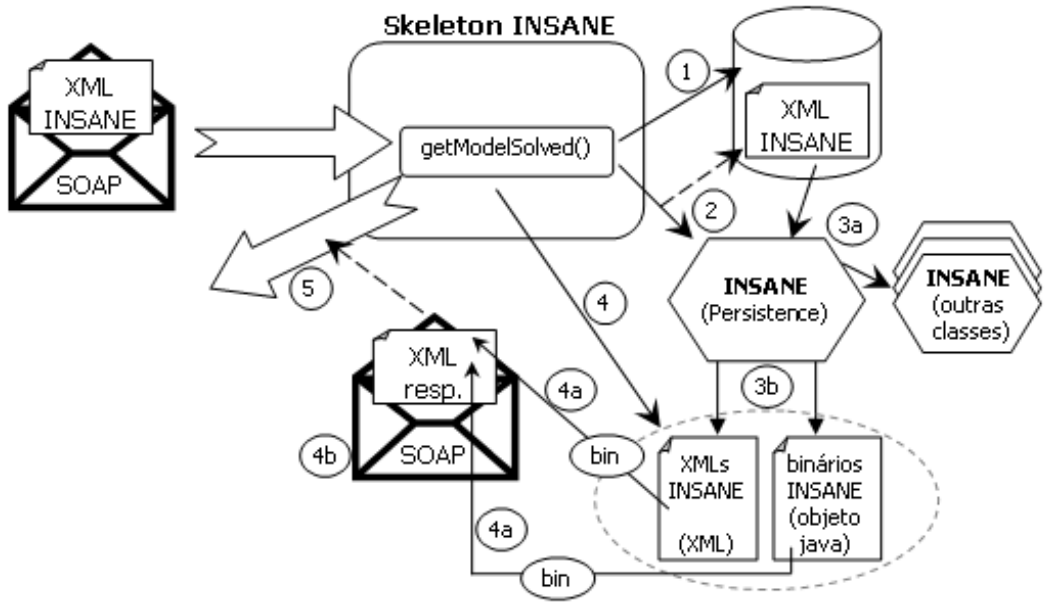


Figura 6.7: *Skeleton* do Serviço Web **INSANE**.

exceção (erro) é emitida, caso, por exemplo, o arquivo *XML* seja inválido, pois durante o processamento é realizada a validação dos dados com o *XML Schema* do **INSANE** (mostrado no Apêndice D), ou haja algum erro durante o processamento;

4. Monta o objeto de resposta para o cliente (4b), que é um elemento *XML* contendo um sub-elemento para cada arquivo gerado, convertido em *bytes* (4a) com o auxílio do *AXIOM* (ou contendo uma mensagem de erro explicitando a exceção ocorrida) inseridos em uma mensagem *SOAP*;

5. Envia a mensagem de resposta ao cliente.

6.5 Disponibilização

Nesta fase, o Serviço Web **INSANE** é disponibilizado para o mundo em um contêiner Web instalado em um servidor, que é um computador conectado à Internet.

O Serviço Web **INSANE**, que será denominado de agora em diante de ***InsaneService***, foi instalado no Servidor **INSANE**, localizado no laboratório do projeto (*InsaneLab*), no Departamento de Engenharia de Estruturas (DEES) da Escola de Engenharia da UFMG. O

servidor é uma computador com processador Intel Core 2 Duo, com 2GB de memória RAM e um disco rígido de 160MB e operando com o sistema Linux Fedora, versão 7.0 (Hat, 2007). O contêiner Web escolhido e instalado nesta máquina foi o Apache *Tomcat* versão 6.0.13 (Tomcat, 2007) que é executado com a máquina virtual Java versão 1.6 (Sun, 2007).

O Apache *Axis*2 versão 1.2 (Axis2, 2007) foi instalado no *Tomcat*, conforme ilustrado na Figura 4.8 do Capítulo 4, para receber o ***InsaneService***.

O Maven (Maven, 2007) foi utilizado para a automação do projeto (compilação e geração do arquivo de *deploy*) conforme explicado no Apêndice C. A Figura 6.8 ilustra este processo.

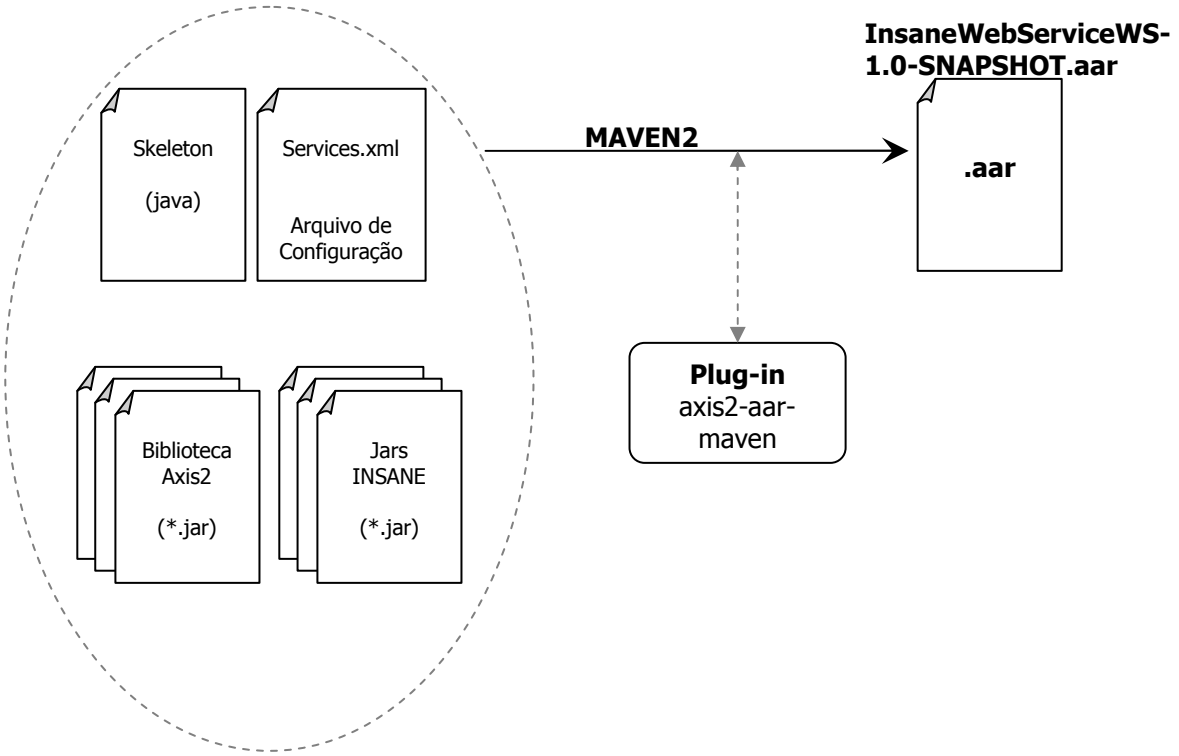


Figura 6.8: Geração do arquivo de *deploy* (.aar) do ***InsaneService***.

Conforme explicado na seção 4.7.2, o *deploy* do ***InsaneService*** é feito utilizando-se a interface Web do *Axis*2 (*Upload* Service). A utilização deste recurso está protegida por senha, exigida pela aplicação *Axis*2, de conhecimento apenas da equipe **INSANE**. A Figura 6.9 mostra a página do *Axis*2 que lista os Serviços Web disponíveis, confirmando a correta instalação do ***InsaneService***.

Figura 6.9: Confirmação da instalação (`deploy`) do ***InsaneService***.

Esta página pode ser acessada pelo endereço `http://insane.dees.ufmg.br:8080/axis2/services/listServices`. Observa-se nesta página, o endereço do ***InsaneService*** (que um futuro cliente precisará conhecer), referenciado como "*Service EPR*" e a lista dos métodos disponíveis para este serviço (*Available Operations*). O título do Serviço Web, ***InsaneService***, é um *link* para exibição do descritor *WSDL* do mesmo.

6.6 Operação

A partir deste momento, o Serviço Web **INSANE** pode ser utilizado por qualquer cliente. O Capítulo 7 mostra três exemplos de programas clientes que utilizam o ***InsaneService***.

6.7 Futuras atualizações

Como mostrado na seção 6.5, a disponibilização do Serviço Web, compreendendo a geração do arquivo de *deploy* (.aar) e instalação (*deploy*) no contêiner Web, é muito simples e observa-se, na prática, que demanda apenas alguns minutos.

Se as classes Java do programa **INSANE** forem atualizadas ou expandidas, basta que a fase acima mencionada seja refeita para que, automaticamente, o ***InsaneService*** esteja atualizado. Esta característica é bastante apropriada ao projeto **INSANE**, por ser um sistema em constante desenvolvimento.

Capítulo 7

EXEMPLOS DE CONSUMIDORES DO INSANESERVICE

7.1 Introdução

No Capítulo 6 foi apresentado e discutido o Serviço Web **INSANE** (***InsaneService***), fruto deste trabalho de dissertação de mestrado.

Como qualquer Serviço Web, ele pode ser consumido por diversos programas clientes, implementados em diferentes linguagens de programação e em diferentes plataformas (ver seção 3.2). Estes clientes podem ser aplicações Web, interfaces gráficas de pré e pós-processamento para "*Desktop*[1]", aplicações para dispositivos microeletrônicos (*palms* e celulares), dentre outros, conforme ilustra a Figura 7.1.

Este capítulo, com o intuito de demonstrar o uso deste serviço, apresentará três diferentes clientes. Além disso, será discutida a geração das classes do Serviço Web do lado do cliente.

É importante ressaltar que todos os clientes foram desenvolvidos apenas para demonstrar o uso do ***InsaneService*** e, por isso, não houve a preocupação com a criação do arquivo de dados *XML*, visto que a geração do mesmo deve ocorrer em um pré-processador apropriado, o que não faz parte do escopo deste trabalho de dissertação. Porém, com o auxílio do *XML Schema* **INSANE** apresentado no Apêndice D e um conhecimento básico da linguagem *XML*, é possível construir o arquivo de dados *XML*, utilizando-se um editor de texto qualquer.

[1]Desktop é a denominação dada ao computador de mesa.

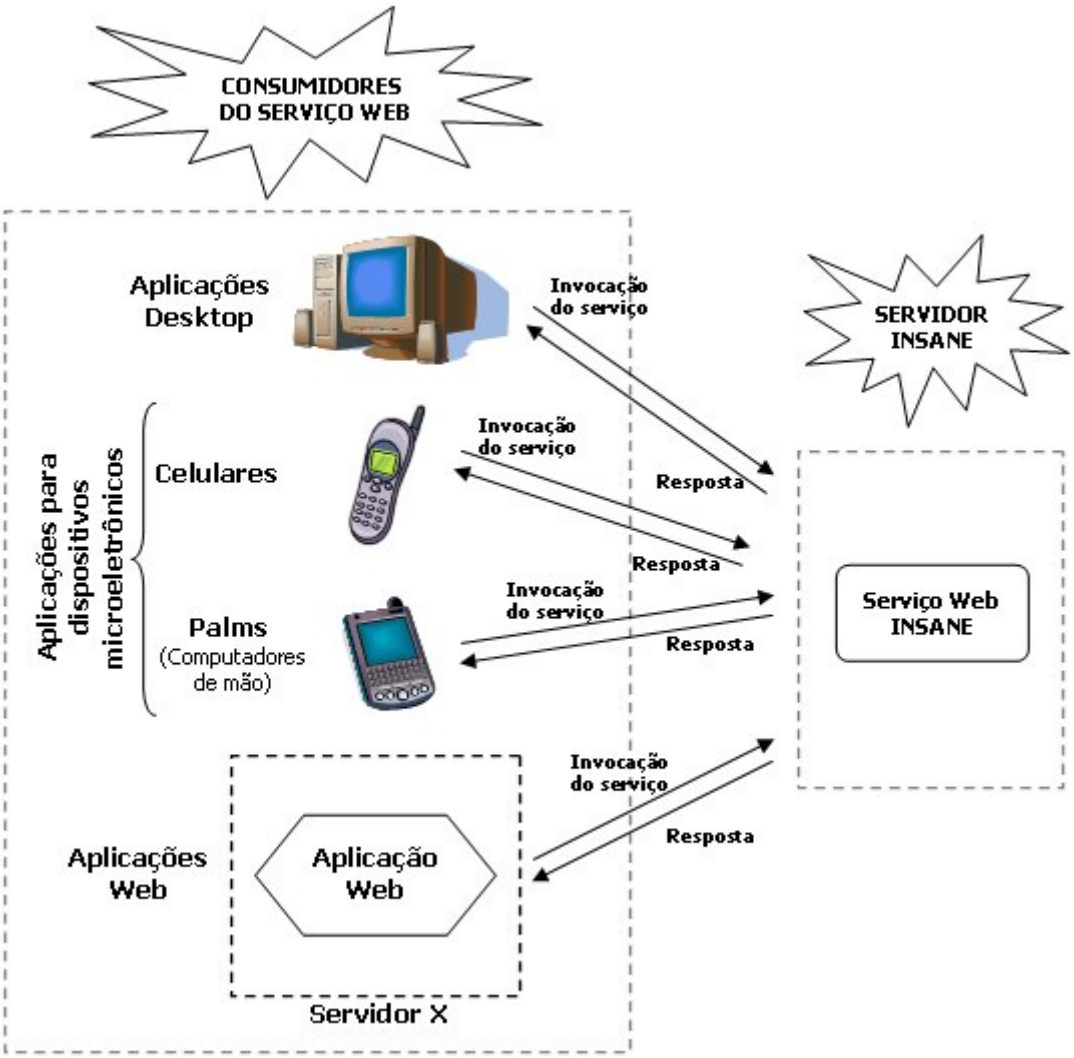


Figura 7.1: Consumidores do Serviço Web **INSANE**.

7.2 Geração das Classes Stub

Neste trabalho, conforme exposto no Capítulo 4, seção 4.7, utilizou-se a plataforma *Axis*2 para a linguagem Java, devido às grandes facilidades oferecidas por ela. Assim, para geração das classes Java do lado do cliente, necessárias à utilização do ***InsaneService*** (*Stub*), o *plug-in* "*Axis*2 *Eclipse Codegen*", disponível em `http://ws.apache.org/axis2/tools/index.html`, foi utilizado. O processo é semelhante ao apresentado na seção 6.4 para geração das classes do Serviço Web e está ilustrado na Figura 7.2.

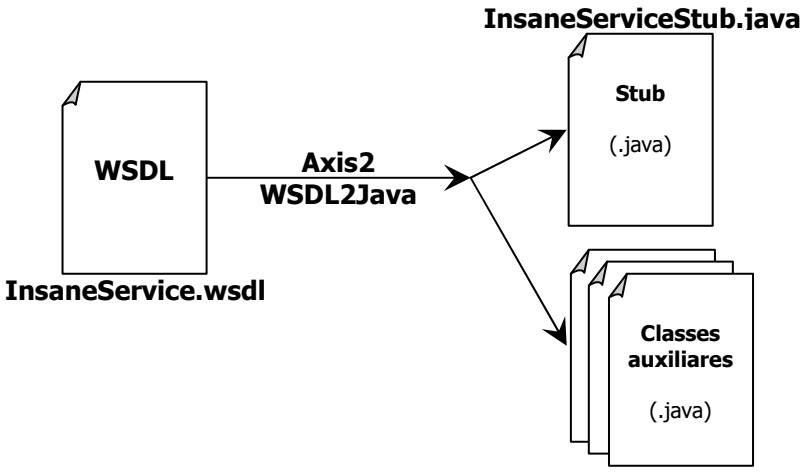


Figura 7.2: Geração das classes Java para uso do Serviço Web **INSANE** (*Stub*).

O arquivo *WSDL*, descritor do Serviço Web **INSANE**, pode ser obtido no endereço Web `http://insane.dees.ufmg.br:8080/axis2/services/InsaneService?wsdl` e é mostrado no Apêndice F. Ele é necessário para a geração do *Stub* e classes auxiliares, como mostrado na Figura 7.2. Estas classes estão prontas para serem usadas nas aplicações clientes, bastando serem copiadas para o diretório de desenvolvimento das mesmas.

7.3 Cliente Java Padrão

O primeiro cliente a ser apresentado é uma aplicação muito simples, constituída por um programa escrito em Java, sem nenhuma interface gráfica com o usuário. A interação com o usuário é feita através de um terminal de linha de comando, no qual o usuário deverá digitar o nome do arquivo *XML* com os dados do modelo **INSANE**.

A *API AXIOM* (ver seção 4.7.4) foi utilizada para manipulação de arquivos *XML*, pois é necessário ler o arquivo fornecido pelo usuário e montar a requisição do Serviço Web, assim como é necessário ler a resposta para obter os arquivos de resultados.

O funcionamento desta aplicação está ilustrado na Figura 7.3.

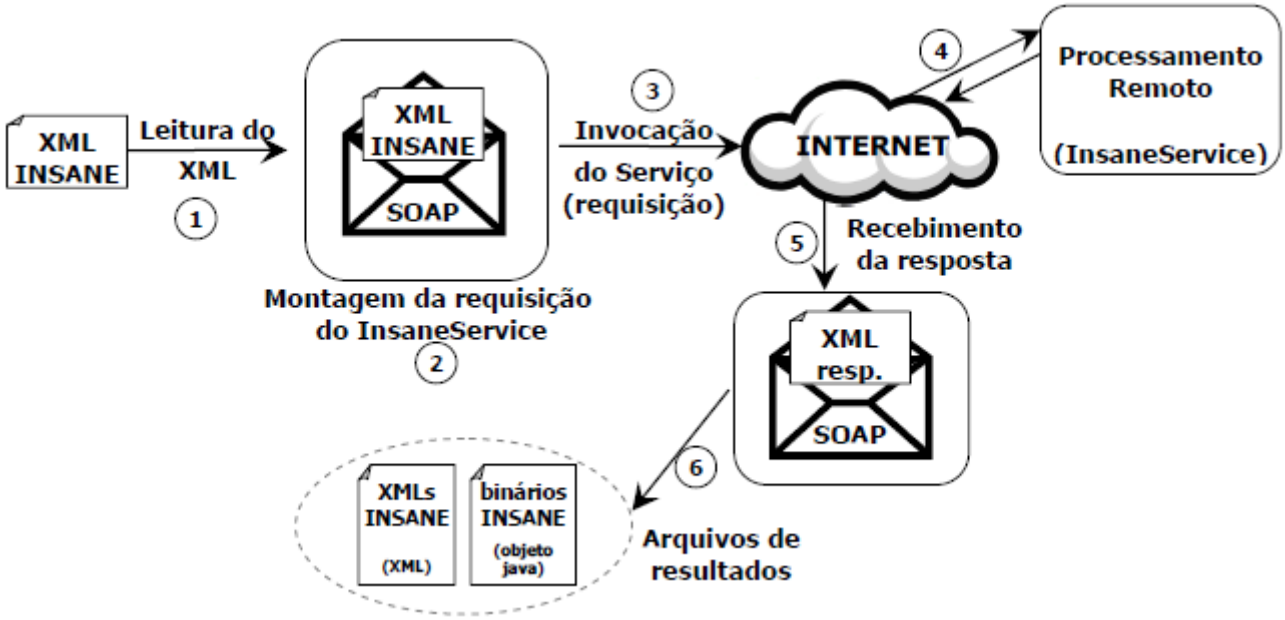


Figura 7.3: Esquema de funcionamento do cliente Java padrão do ***InsaneService***.

A aplicação lê o nome do arquivo *XML* de dados digitado pelo usuário (1), monta a requisição do ***InsaneService*** usando a classe *Stub* (gerada conforme mostrado na seção 7.2) passando os dados do arquivo *XML* como parâmetro, a qual é a responsável por gerar a mensagem *SOAP* e inserir os dados *XML* (2). A seguir é realizada a invocação do Serviço Web (3) através da Internet. No Servidor é realizado o processamento da solicitação (4), conforme exposto na seção 6.4 e mostrado nas Figuras 6.4 e 6.7 e a resposta é recebida pela aplicação cliente (5). Os arquivos de resultados são, então, extraídos e salvos localmente no disco rígido do computador onde a aplicação está sendo executada (6).

O código Java completo deste cliente, pode ser encontrado no endereço Web: `http://insane.dees.ufmg.br:8080/InsaneWeb/app?page=Client&service=page`. Nele, vale ressaltar a criação da instância do *Stub* (objeto *stub*), e da requisição (objeto req):

```
br.ufmg.dees.insane.webservice.InsaneServiceStub stub = new
    br.ufmg.dees.insane.webservice.InsaneServiceStub
```

```
    ("http://insane.dees.ufmg.br:8080/axis2/services/InsaneService" );
    .... ....
 br.ufmg.dees.insane.webservice.InsaneServiceStub.GetModelSolved req= new
    br.ufmg.dees.insane.webservice.InsaneServiceStub.GetModelSolved();
```

Os parâmetros da requisição (*XML*) são atribuídos ao objeto de requisição nas linhas:

```
OMElement elem2 =  AxiomUtils.readXmlFile(fileName);
    ... ....
req.setParam0(elem2);
    ... ....
req.setParam1(fileName);
```

Além disso, a invocação do serviço acontece na linha:

```
br.ufmg.dees.insane.webservice.InsaneServiceStub.GetModelSolvedResponse res =
            stub.getModelSolved(req);
```

O restante do código extrai os objetos de resposta e salva os arquivos localmente.

A Figura 7.4 mostra a aplicação, denominada "*SimpleClient*" em execução, durante a qual é fornecido, como arquivo de dados, o *XML* apresentado no Apêndice E.

Na última linha da janela de console, observa-se que todo o processou dispendeu 10 segundos.

Finalmente, as Figuras 7.5 e 7.6 mostram os arquivos de resultados recebidos e salvos na máquina da aplicação cliente, sendo que primeira mostra o arquivo *XML* e a segunda o arquivo ISN (objeto Java **INSANE**).

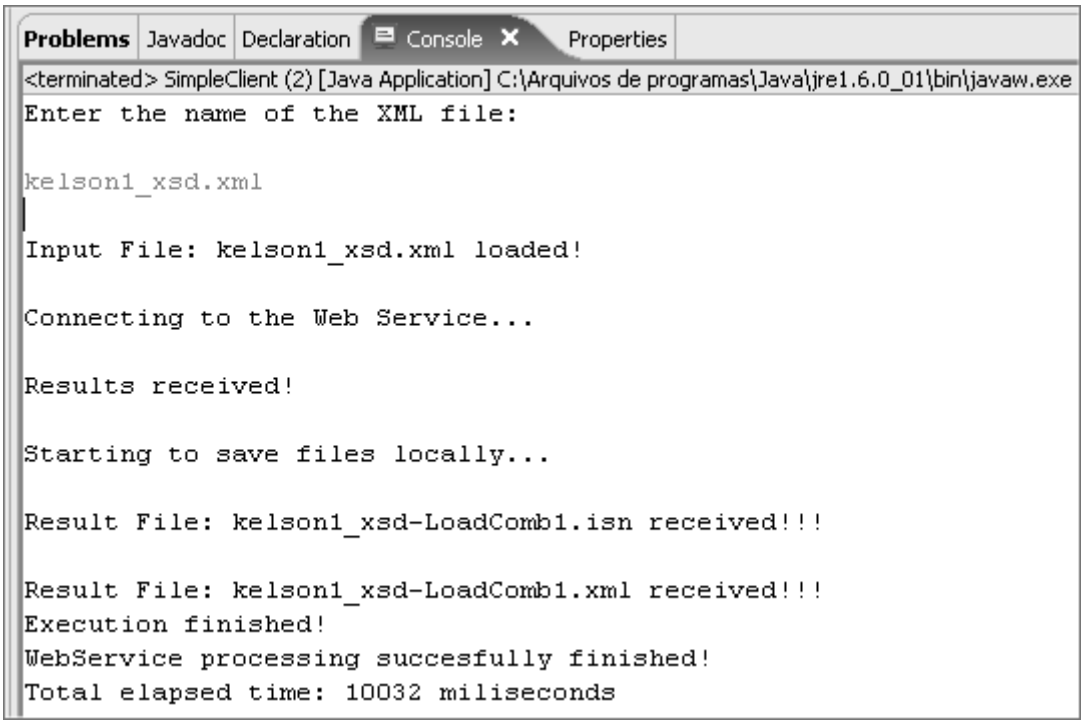


Figura 7.4: Cliente Java padrão "*SimpleClient*" em execução.

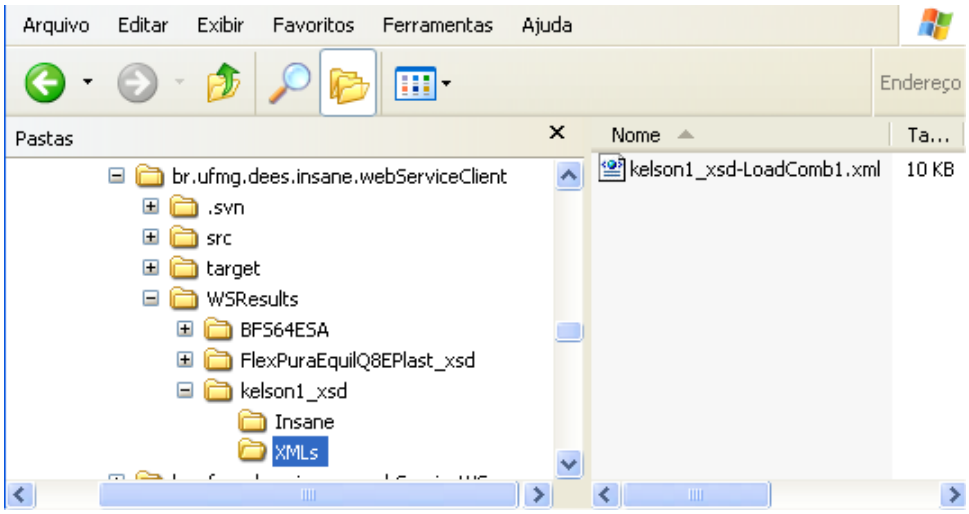


Figura 7.5: Arquivo *XML* de resultado recebido.

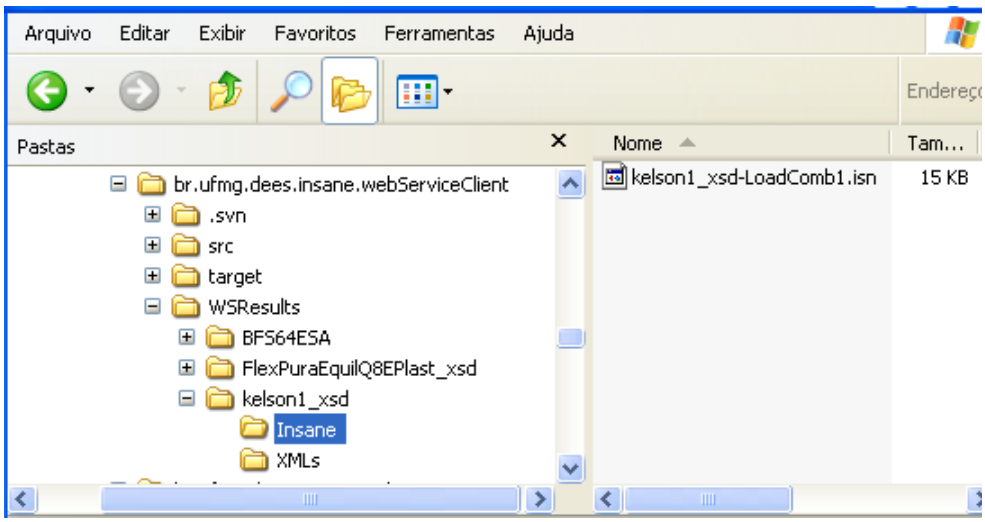


Figura 7.6: Arquivo ISN (objeto Java **INSANE**) de resultado recebido.

7.4 Cliente Java Swing

O sistema **INSANE** possui, em desenvolvimento, uma aplicação pré-processadora, processadora e pós-processadora, que contemplará os antigos e novos trabalhos do projeto e é baseada na *API* Java *Swing*, que oferece poderosos recursos de computação gráfica. Futuramente, será lançada uma versão dessa aplicação para instalação e execução em computadores de mesa (*Desktop*).

Foi adicionada na parte da aplicação pré-processadora uma nova funcionalidade que permite a escolha da realização do processamento (solução do modelo) remotamente, utilizando o ***InsaneService***, conforme mostrado na Figura 7.7.

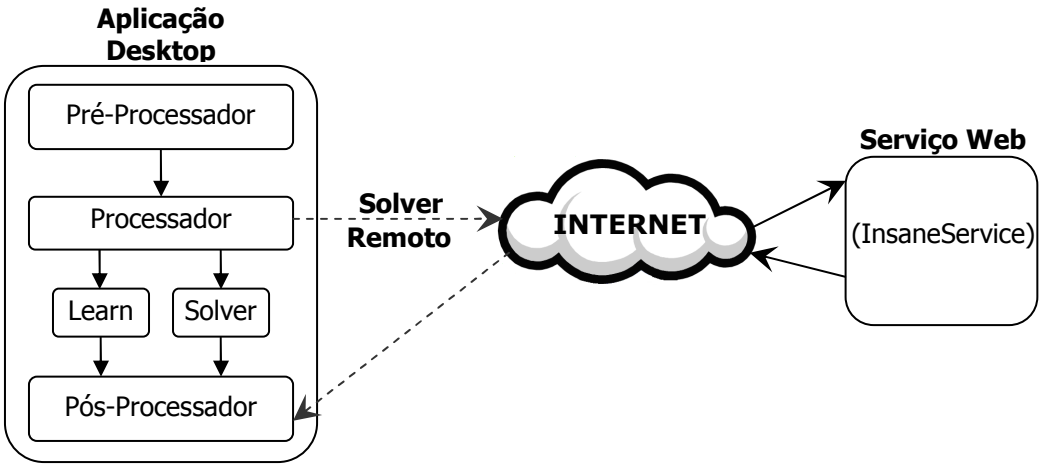


Figura 7.7: Esquema de funcionamento do cliente Java *Swing* para o ***InsaneService***.

A vantagem desta opção é permitir ao usuário que tem a aplicação instalada em seu computador, usá-la para geração do modelo (dados) e realizar o processamento remoto, que pode já estar atualizado com alguma nova funcionalidade, e visualizar os resultados com o seu poderoso pós-processador, desenvolvido por Penna (2007), sem ter que obter a nova versão do programa e refazer a sua instalação.

A implementação desta funcionalidade se dá através da criação de uma classe para invocação do serviço remoto, cujo código é semelhante ao utilizado na aplicação Java padrão (seção 7.5), apenas alterando-se a forma de interação com o usuário para a obtenção do arquivo de dados. Neste caso, utiliza-se caixas de diálogo *Swing* (*API* `javax.swing`).

A Figura 7.8 ilustra a utilização da escolha de processamento remoto na aplicação Java *Swing*, acima discutida.

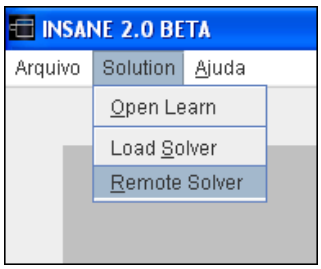


Figura 7.8: Escolha do processamento remoto no cliente Java *Swing* **INSANE**.

Ao escolher esta opção, abre-se uma janela para escolha do arquivo de dados *XML*, como mostrado na Figura 7.9.

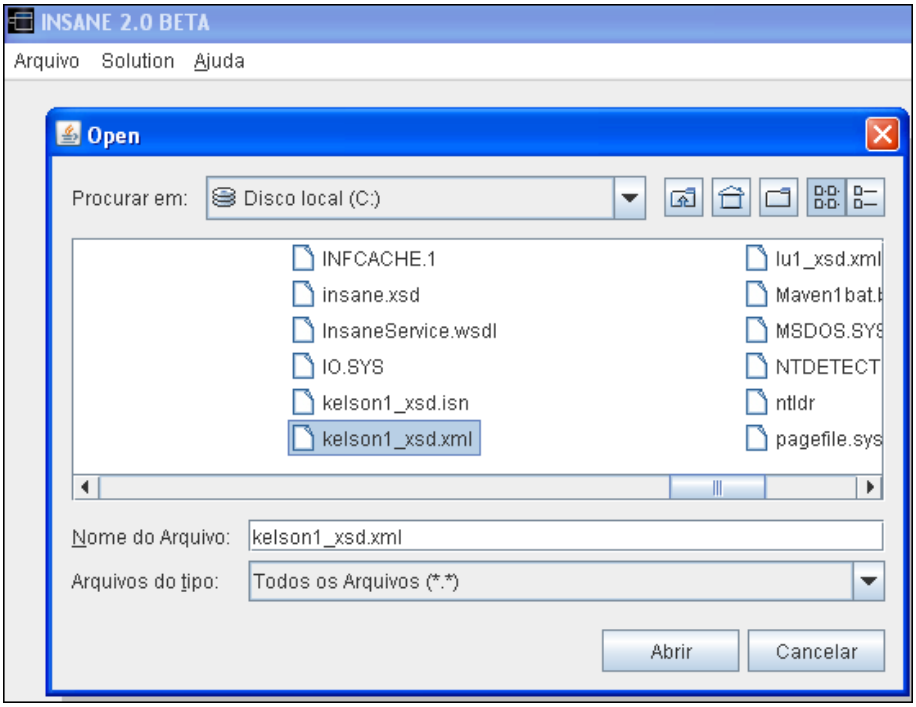


Figura 7.9: Escolha do arquivo de dados *XML* no cliente Java *Swing* **INSANE**.

O esquema de funcionamento da invocação e recebimento dos resultados desta opção de solução de modelos **INSANE**, via processamento remoto, acontece do mesmo modo ilustrado na Figura 7.3. O cliente agora é uma aplicação gráfica, mas o funcionamento permanece o mesmo.

A Figura 7.10 mostra a mensagem de sucesso após o processamento escolhido.

Figura 7.10: Mensagem de sucesso do processamento remoto no cliente Java *Swing* **INSANE**.

Depois do processamento concluído, pode-se visualizar os resultados, interativamente, no módulo de pós-processamento da aplicação. É possível escolher as grandezas de resultados, as cores dos elementos, escala, elementos exibidos, entre várias outras opções. A Figura 7.11 mostra um exemplo dessa visualização, onde são exibidas as variações dos deslocamentos verticais (Dy) e das tensões horizontais (Sigma XX) do modelo de estado plano de tensões referente ao arquivo `kelson1_xsd.xml`.

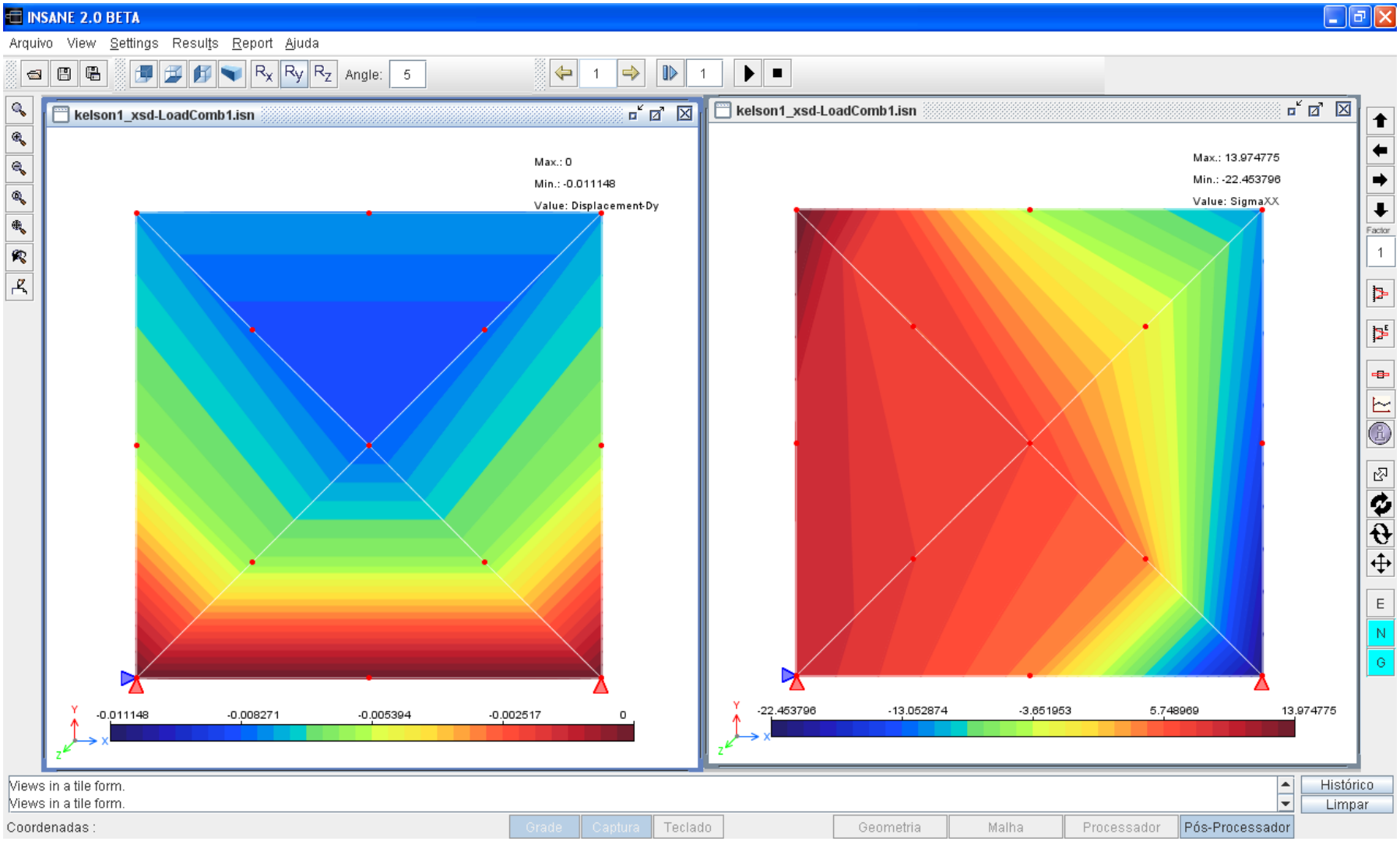


Figura 7.11: Visualização de resultados no cliente Java *Swing* **INSANE**.

7.5 Cliente Web

O terceiro cliente escolhido para demonstração é o mais interessante de todos, pois permitirá o uso do Serviço Web por qualquer usuário no mundo, sem a necessidade de instalação de nenhum programa. Bastará o uso de um navegador Web funcionando em um computador com acesso à Internet. Este cliente é uma aplicação Web, denominada ***InsaneWeb***.

Conforme explicado na seção 5.1 uma Aplicação Web se refere a um programa que é executado pela Web, tipicamente através de um navegador Web e oferece interação com o usuário. O usuário acessa o *site* da aplicação, a qual está instalada fisicamente em um servidor localizado em algum ponto da Internet, e passa a interagir com a mesma. A Figura 7.12 ilustra este processo.

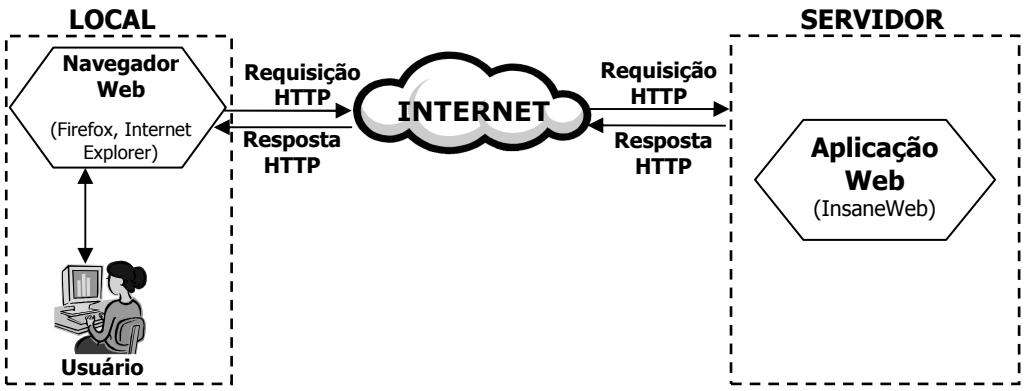


Figura 7.12: Esquema de funcionamento de uma aplicação Web típica.

O ***InsaneWeb*** é uma aplicação Web, constituída por um conjunto de arquivos *HTML*, figuras, *servlet*s e folhas de estilo *CSS* que são utilizadas através de navegadores Web comuns (*Firefox*, *Netscape*, *Internet Explorer*, entre outros) e o usuário só precisa de uma conexão ativa com a Internet. Nenhum outro programa é necessário para a sua utilização.

Para o desenvolvimento desta aplicação, foram utilizadas todas as tecnologias apresentadas no Capítulo 5. O *Tapestry* (ver seção 5.4) foi utilizado para geração de conteúdo dinâmico e o *Axis* (ver seção 4.7) para encapsulamento das operações com o Serviço Web. Foram geradas as classes *Stub*, do lado do cliente, exatamente da mesma forma, como mostrado para os clientes Java padrão e *Swing*.

Foi criado um novo projeto **INSANE**, também desenvolvido em Java, denominado "`br.`

`ufmg.dees.insane.ui.web`", com características próprias de uma aplicação web, ou seja, além de classes Java presentes em qualquer projeto Java comum, há também vários recursos próprios para Web. A Figura 7.13 mostra a estrutura de diretórios deste novo projeto.

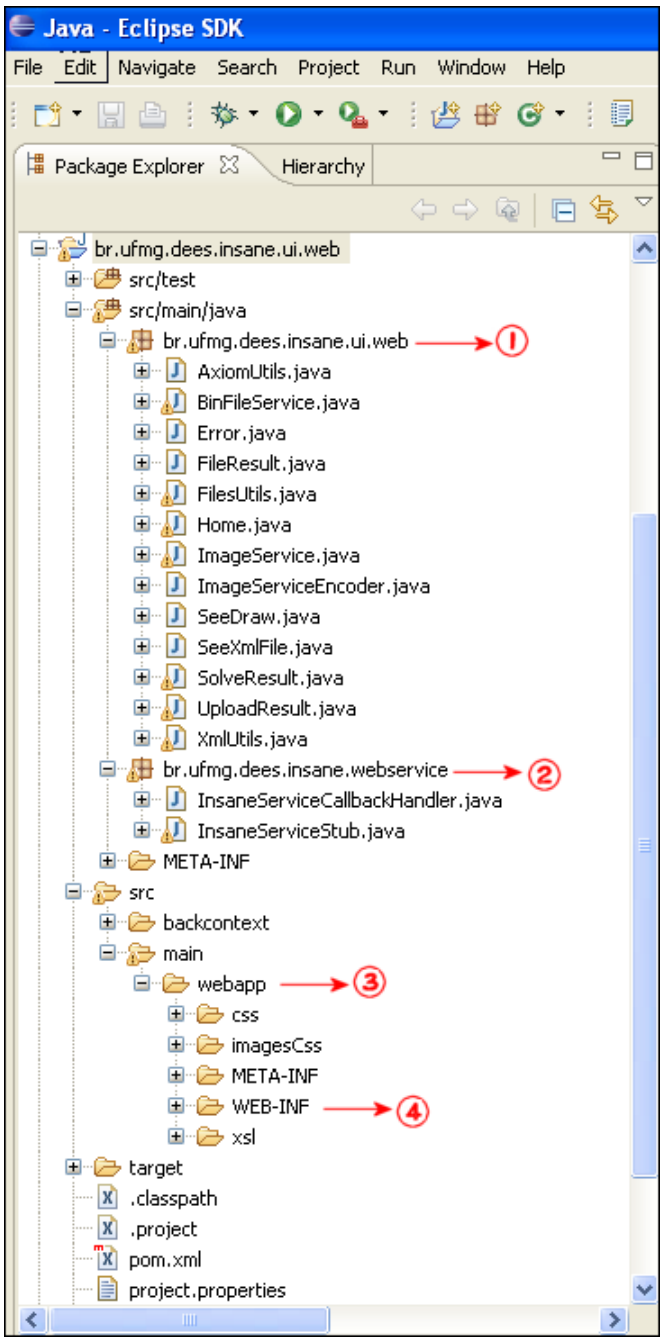


Figura 7.13: Estrutura de diretórios do projeto *br.ufmg.dees.insane.ui.web*.

Como mostrado na figura, no diretório "`src/main/java`" ficam armazenadas as classes Java. Neste, observa-se o pacote `br.ufmg.dees.insane.ui.web` (1) que contém classes utilizadas pelo *Tapestry*, exclusivas da aplicação Web. O pacote `br.ufmg.dees.insane.webservice` (2) contém as classes *Stub* do Serviço Web. O diretório "`src/main/webapp`" (3) é o local onde são armazenados os arquivos que são efetivamente empacotados para instalação no contêiner Web. Nele, destaca-se o diretório "`WEB-INF`" (4), que contém os arquivos *HTML* e `PAGE` (que guarda informações para o *Tapestry*) além do arquivo "`web.xml`", que é o descritor da aplicação. Neste diretório também são copiados os arquivos "`.class`" gerados na compilação e todos as bibliotecas de terceiros necessárias para a execução da aplicação (arquivos "`.jar`"). Há ainda outros diretórios, como o "`META-INF`", que contém o arquivo "`context.xml`" com outras configurações específicas, o "`css`", que guarda as folhas de estilo, o "`imagesCss`", onde ficam as imagens utilizadas pelas folhas de estilo e o "`xsl`", de armazenamento da folha de estilo de transformação para dados *XML*.

Uma aplicação Web é instalada em um contêiner Web para que fique disponível para uso através da Internet. Esta instalação pode ser feita através de um arquivo de *Deploy*, de extensão "`.war`" (*Web Archive*). Com o auxílio do Maven (ver o Apêndice C), a geração deste arquivo ocorre de maneira muito simples, como mostrado na Figura 7.14.

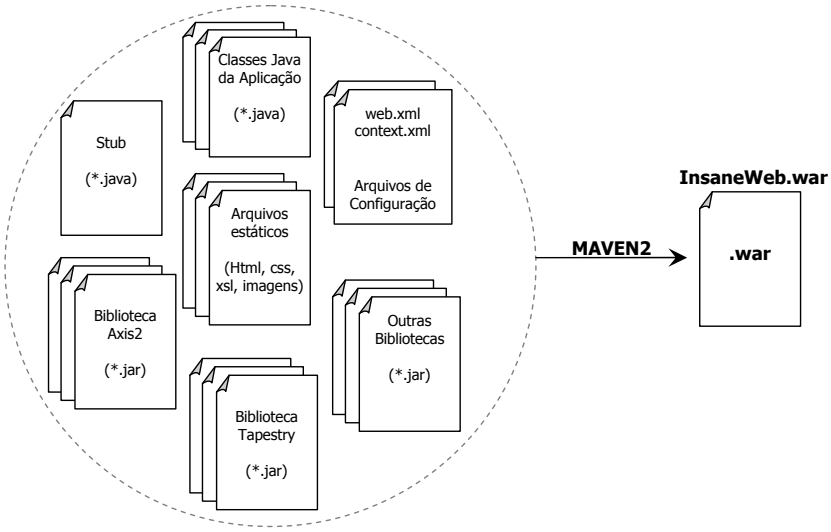


Figura 7.14: Geração do arquivo de *Deploy* da aplicação Web (***InsaneWeb***).

É importante observar, que este cliente não acessa e não referencia nenhuma classe do

INSANE, ou seja, nenhum arquivo ".jar" dos projetos **INSANE** está presente neste projeto. Isto demonstra que toda a interação com as classes do núcleo numérico **INSANE** está confinada no Serviço Web. Esta característica é muito importante, pois mostra que qualquer cliente pode usufruir do ***InsaneService*** sem ter que conhecer a sua implementação ou adquirir os seus arquivos "`.jar`".

A instalação da aplicação no contêiner Web (*`Tomcat`*) pode ser feita de duas maneiras: copiando-se o arquivo para o seu diretório específico, *webapps* (neste caso é necessário reiniciar o contêiner para que ele possa localizar e instalar a aplicação), ou usando o interface Web na sua página de administração, como se pode ver na Figura 7.15.

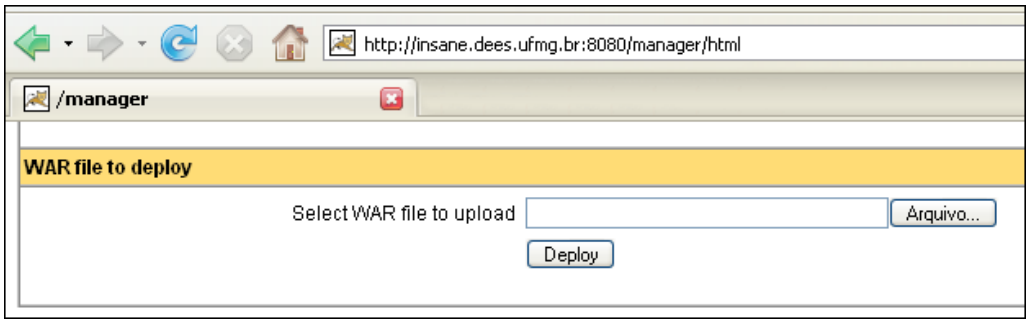


Figura 7.15: *`Deploy`* de aplicações Web no *`Tomcat`*.

O esquema de funcionamento do ***InsaneWeb***, incluindo o processo de consumo do ***InsaneService***, está ilustrado na Figura 7.16.

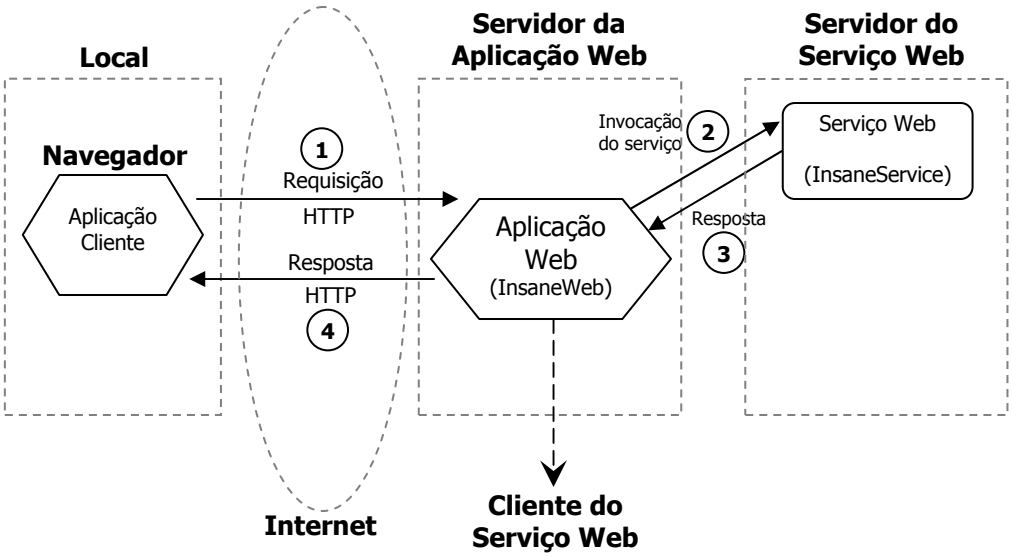


Figura 7.16: Esquema de funcionamento do cliente Web (***InsaneWeb***) do ***InsaneService***.

A aplicação cliente, no caso um usuário usando um navegador Web, acessa o "*site*" da aplicação ***InsaneWeb***, fornece o arquivo de dados *XML* e, ao solicitar a resolução do problema (1), a aplicação Web invoca o ***InsaneService*** através do envio de mensagens *SOAP* contendo os dados do modelo em formato *XML* (2). O Serviço Web faz o processamento da requisição e retorna a resposta também através do envio de mensagens *SOAP* (3). A aplicação Web processa esta resposta e envia uma resposta ao usuário, exibindo-a em forma de páginas *HTML* ou oferecendo o descarregamento dos arquivos de resultados, conforme a opção do usuário (4).

O endereço Web desta aplicação é: `http://insane.dees.ufmg.br:8080/InsaneWeb/app`. A Figura 7.17 mostra a página principal do ***InsaneWeb***.

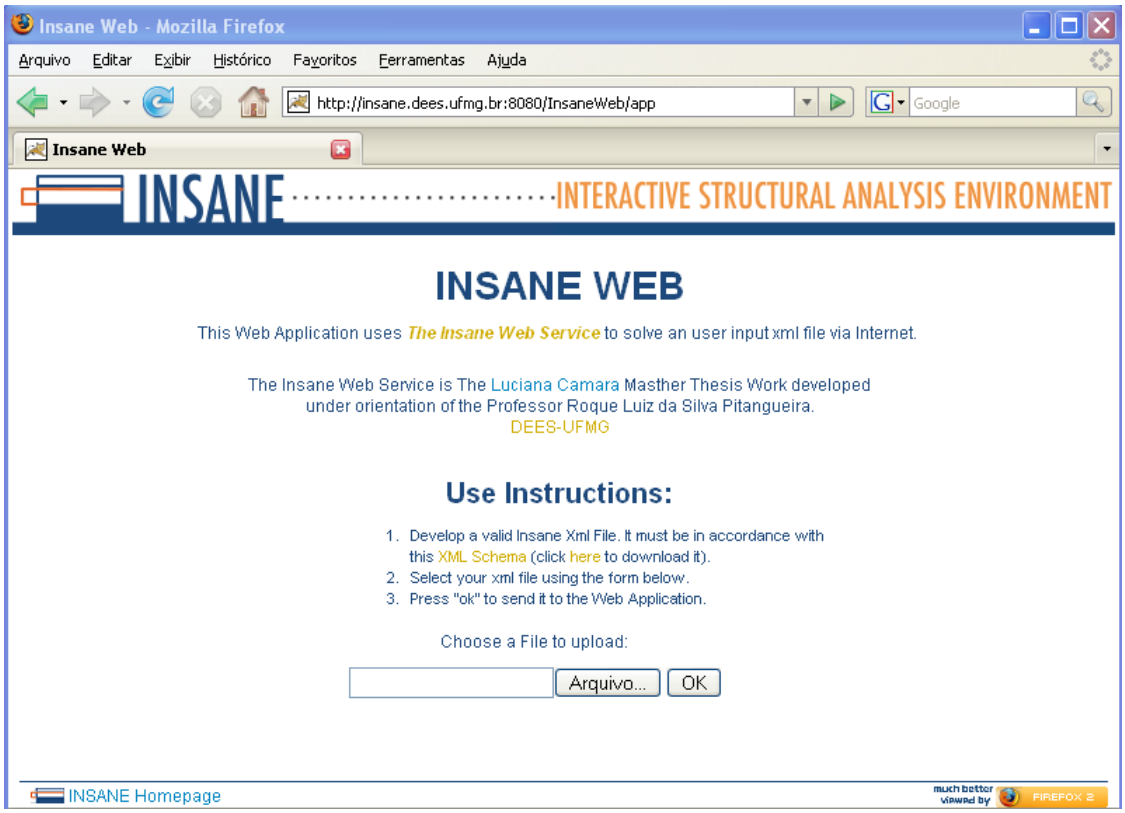


Figura 7.17: Página principal do ***InsaneWeb***.

Nesta página, o usuário tem acesso a algumas informações sobre o Serviço Web **INSANE** e o *XML Schema* através de "*links*" e, através da caixa de diálogo "*Choose a File to Upload*", no fim da mesma, ele pode fornecer um arquivo *XML* para envio à aplicação. Na próxima página (Figura 7.18), de título "***InsaneWeb*** - *Upload*", que é exibida ao pressionar-se o botão "OK", o recebimento do arquivo (no lado do servidor da aplicação Web) é informado e o conteúdo do mesmo é exibido numa área de texto.

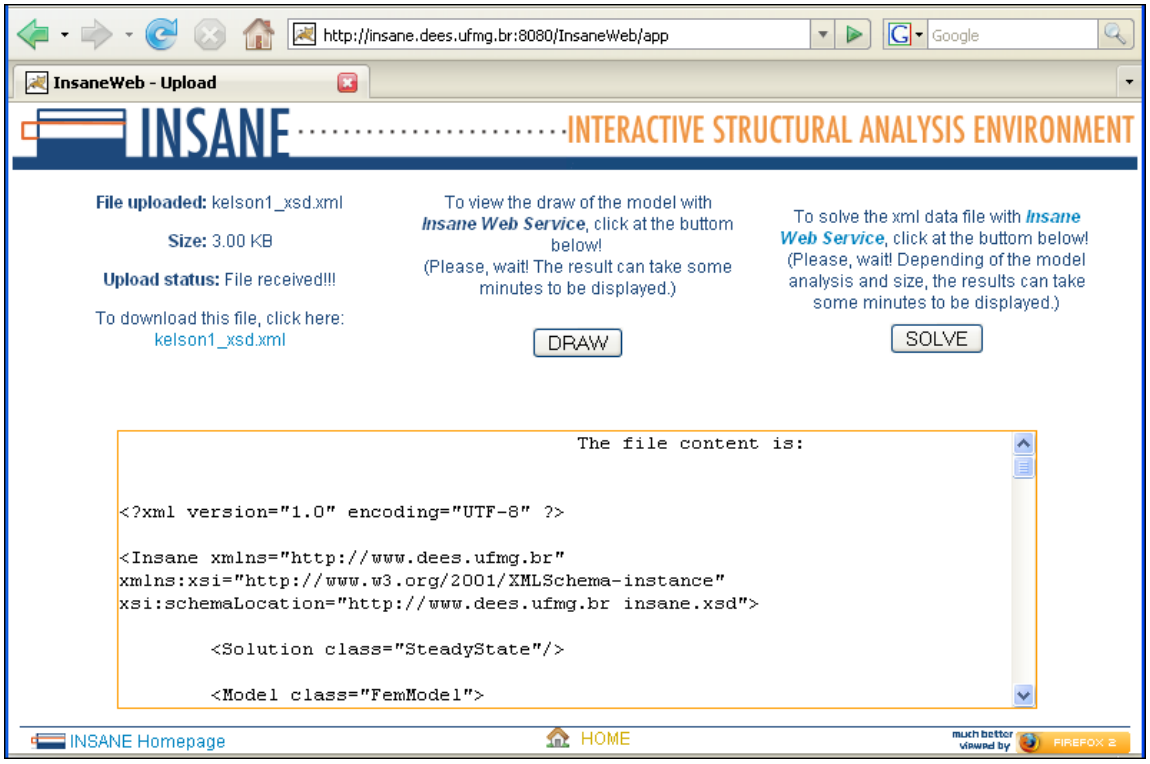


Figura 7.18: Envio do arquivo *XML* ao ***InsaneWeb***.

Neste momento, o arquivo fornecido pelo usuário está no Servidor onde a aplicação ***InsaneWeb*** foi instalada. Nesta página, o usuário pode solicitar a geração do desenho do modelo (ainda não resolvido), pressionando o botão "*DRAW*". Esta ação aciona a invocação de um outro método disponibilizado pelo ***InsaneService*** (seção 6.4), o "*getFigure()*", que gera a figura do modelo a partir do arquivo de dados *XML*. A Figura 7.19 mostra a imagem gerada através desta opção, para o arquivo *XML* de exemplo (mostrado no Apêndice E).

Ainda na página "InsaneWeb - *Upload*", o usuário pode solicitar a solução do modelo através do ***InsaneService***, pressionando o botão "*SOLVE*". Neste momento, a aplicação Web invoca o ***InsaneService***, da mesma maneira descrita para os outros clientes e o usuário

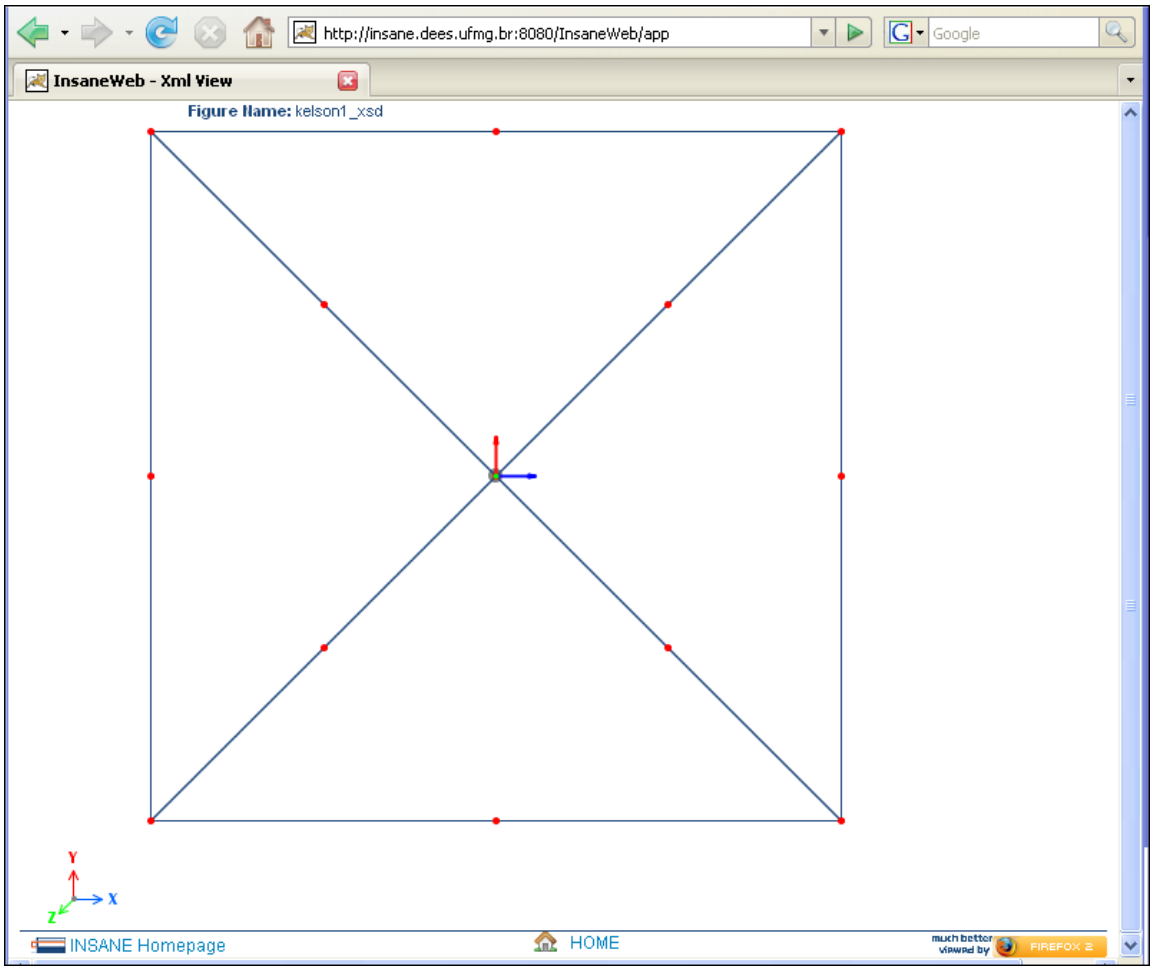


Figura 7.19: Desenho do modelo no ***InsaneWeb***.

é redirecionado para a página de resultados, na qual são exibidas duas listas, relacionando os arquivos gerados no processamento e recebidos pela aplicação ***InsaneWeb***. Esta página está mostrada na Figura 7.20.

O cliente do Serviço Web é a aplicação Web ***InsaneWeb***, logo, os arquivos de resultados foram recebidos e salvos no disco rígido do Servidor onde a aplicação Web está instalada. Nesta página, o usuário tem a opção de copiar os arquivos para o seu computador através do *link* "*download*" apresentado junto à imagem de um disquete. Este processo está ilustrado na Figura 7.21.

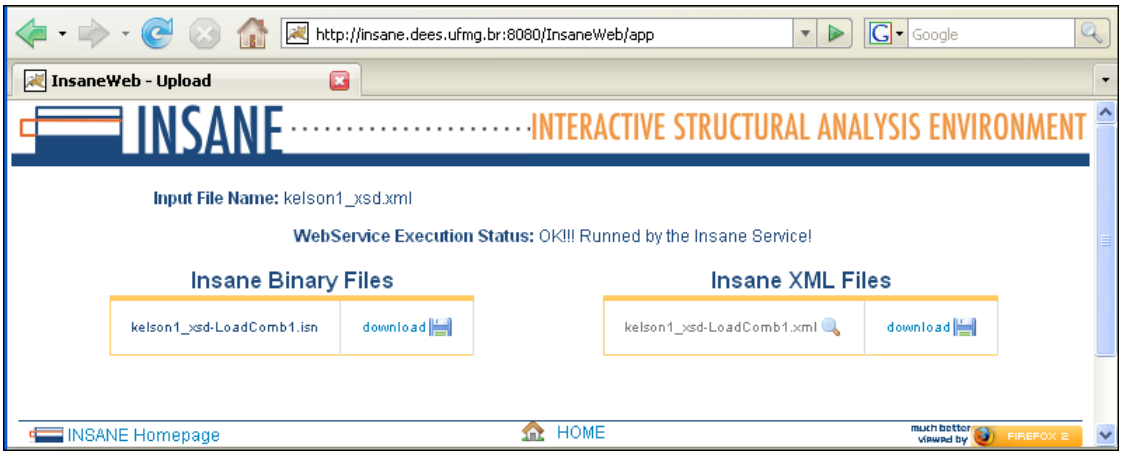


Figura 7.20: Visualização dos resultados do ***InsaneService*** invocados pelo ***InsaneWeb***.

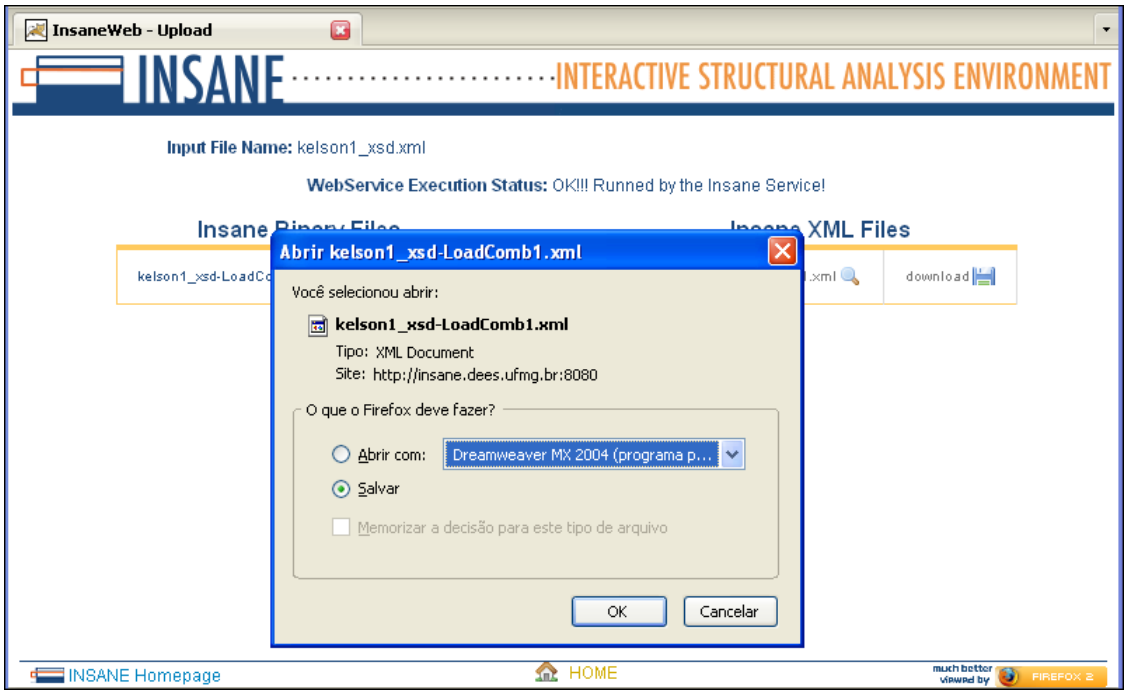


Figura 7.21: *Download* dos arquivos de resultados no ***InsaneWeb***.

Ainda nesta página,o usuário pode visualizar o conteúdo dos arquivos *XML* de resultados em forma de um relatório formatado, de agradável aparência. Para isso, a tecnologia de transformação de *XML*, o *XSLT* (ver seção 5.6) foi usada juntamente com uma folha de estilos *CSS* (ver seção 5.3) específica. A Figura 7.22 mostra uma parte do relatório gerado para este processamento, que é exibido através do *link* com o nome do arquivo de resultados, ao lado da imagem de uma lupa (ver Figura 7.20).

Figura 7.22: Relatório de resultados apresentado no ***InsaneWeb***.

Para a visualização completa do relatório (que é extenso), o leitor é convidado a utilizar a aplicação ***InsaneWeb***, fornecendo o arquivo *XML* do Apêndice E.

Nesta página de visualização do relatório, "*Xml View*", é possível também visualizar imagens dos resultados, através da caixa de diálogo "*Choose a Result to display*" (Figura 7.23). As grandezas de resultados são obtidas através de outro método acrescentado ao ***InsaneService***, o "*getModelKeys()*".

A Figura 7.24 mostra a imagem gerada da grandeza escolhida, no caso, deslocamentos

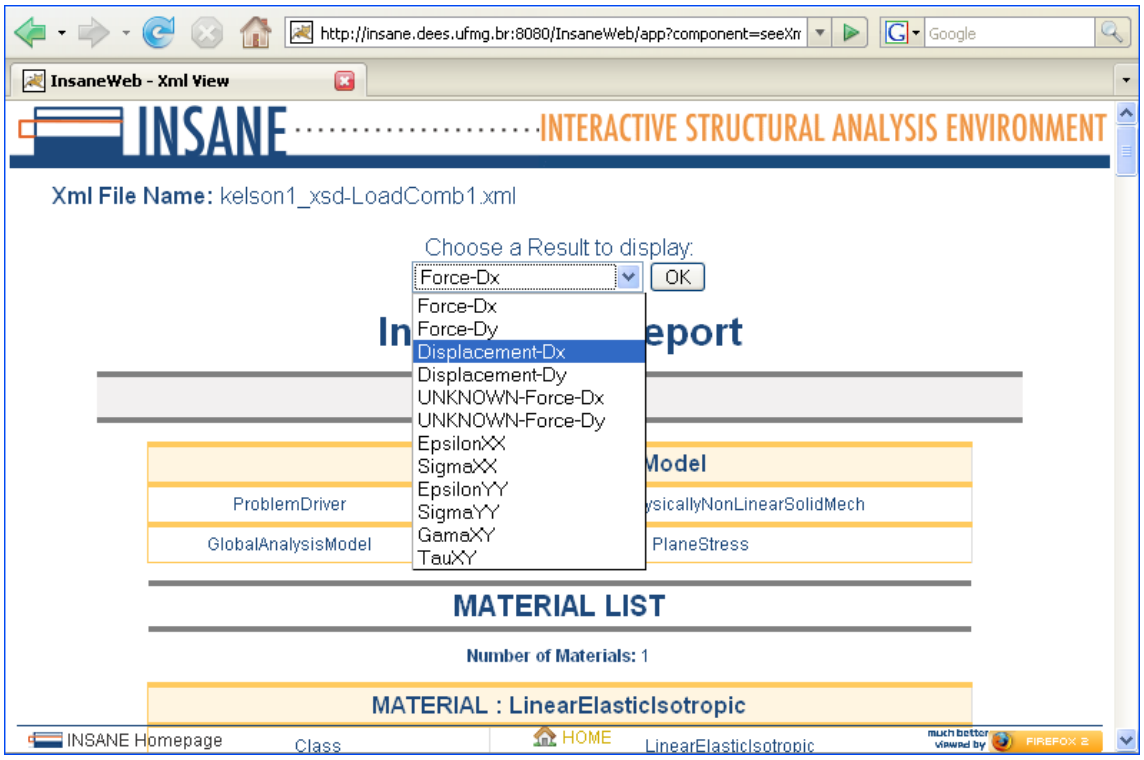


Figura 7.23: Escolha da grandeza de resultado para exibição no ***InsaneWeb***.

horizontais (Dx), que é exibida ao se pressionar o botão "*OK*". Esta figura é obtida com o último método acrescentado ao ***InsaneService***, "*getResultFigure()*". Este método, utiliza uma das classes de geração de imagens pertencentes ao projeto de pós-processamento **INSANE** (Penna, 2007).

A Figura 7.25 mostra a imagem de resultados, deslocamentos transversais (Dz), para um modelo de placa anular, apenas para efeito ilustrativo da aplicação.

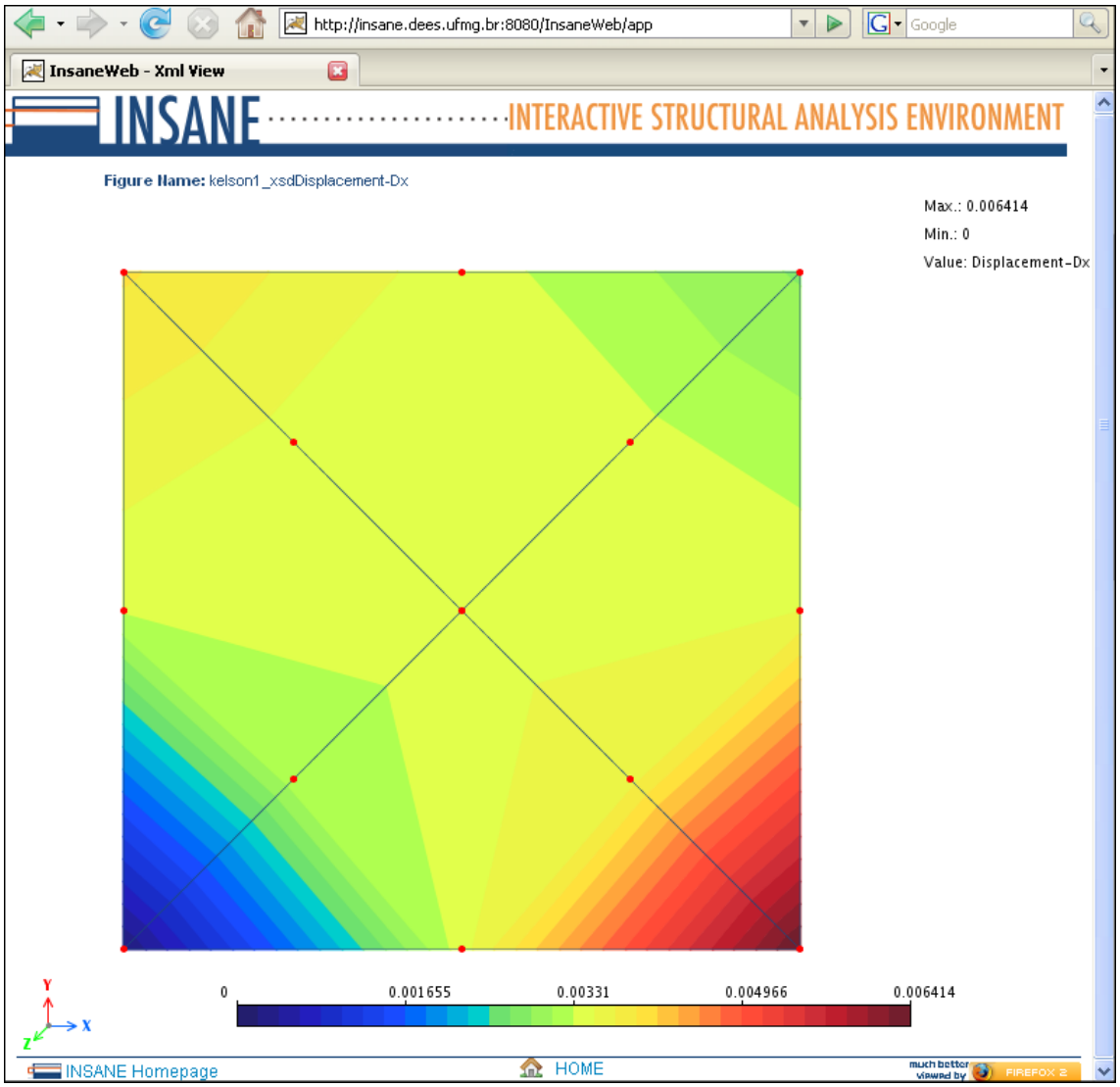


Figura 7.24: Visualização de resultados (1) no ***InsaneWeb***.

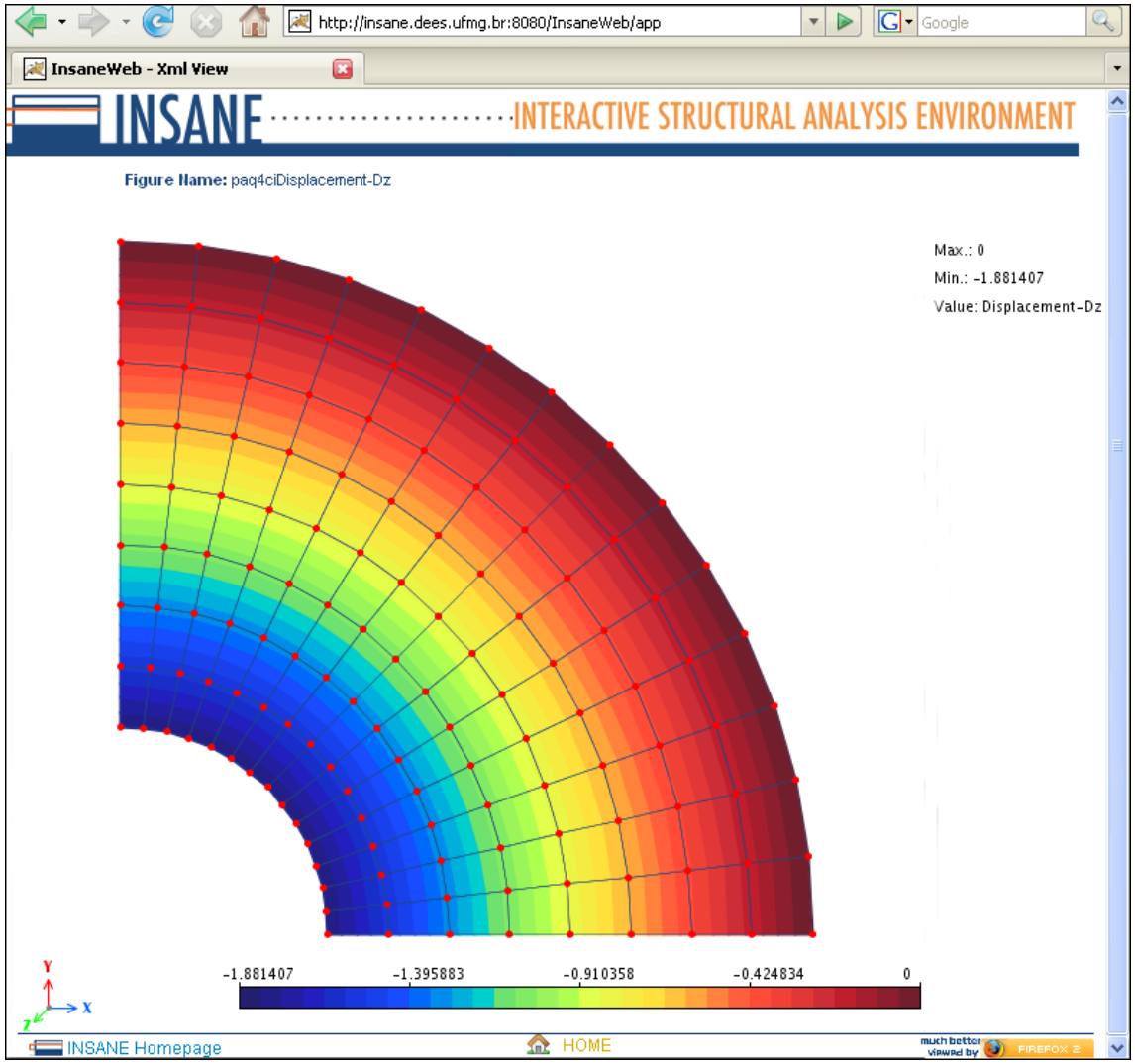


Figura 7.25: Visualização de resultados (2) no ***Insane Web***.

Capítulo 8

CONSIDERAÇÕES FINAIS

O trabalho apresentado se adequa às diretrizes gerais do projeto **INSANE** em relação ao desenvolvimento colaborativo, ao conceito de *software* livre e à pesquisa na área de métodos computacionais para modelos discretos de análise estrutural. Além disso, permite ao **INSANE** usufruir do novo paradigma de comunicação proporcionado pela evolução da Web, integrando ao sistema modernas tecnologias na área de computação distribuída baseada em padrões abertos e públicos.

8.1 Contribuições deste Trabalho

O Serviço Web desenvolvido (***InsaneService***) contribui bastante para as pesquisas na área de métodos numéricos e computacionais pois disponibiliza o núcleo numérico do **INSANE** à comunidade científica.

A utilização de Serviços Web proporciona ao **INSANE** a reutilização de código mesmo quando integrado à sistemas desenvolvidos em outras linguagens e plataformas (interoperabilidade), atualização de seu núcleo numérico de maneira transparente aos seus usuários, o que propicia uma grande flexibilidade e uma interação simples entre clientes e servidores, pois a tecnologia é baseada em padrões simples e públicos.

O cliente Web desenvolvido, ***InsaneWeb***, oferece a possibilidade de uso imediato do ***InsaneService***, através da Internet, pois não exige nenhuma instalação de programas, adaptação ou aprendizado das teorias, tecnologias e códigos de implementação. Além disso, esta aplicação Web, embora não utilize modernos recursos de computação gráfica interativa como no

cliente Java *Swing*, pois as atuais tecnologias para aplicações Web ainda não contam com estes recursos, oferece algumas opções de pós-processamento "*online*", como geração de imagens e relatório de resultados. É importante observar, porém, que já começam a ser discutidas e desenvolvidas, na comunidade Web, tecnologias de geometria computacional para ambientes Web, que permitirão a criação de aplicativos *RIA* ou "*Rich Internet Applications*", que adicionam características e funcionalidades de uma aplicação desktop tradicional à aplicações Web. A solução dos modelos através deste cliente, assim como a geração das imagens, demandou poucos segundos de espera, o que é bastante razoável para um processamento remoto. Para modelos mais complexos, que envolvam análises mais demoradas, este tempo será maior, mas observou-se, através de testes iniciais, um aumento de apenas um minuto em relação ao processamento local (aplicação Java *Swing*). Por ser acessado via Internet, pessoas em qualquer lugar do mundo já podem analisar um modelo discreto de elementos finitos "*online*" e gratuitamente. As facilidades oferecidas pela enorme evolução da Internet chegam finalmente à engenharia brasileira.

O Serviço Web desenvolvido é apenas o início de uma nova fase do sistema **INSANE**. As tecnologias e recursos aprendidos e consolidados oferecem respaldo técnico a futuros trabalhos e aprimoramentos do sistema envolvendo a computação distribuída e aplicativos Web.

8.2 Sugestões para Trabalhos Futuros

A seguir, são listadas algumas idéias para trabalhos futuros.

1. Aprimoramento do Serviço Web, visando aumento de desempenho com a implementação de processamento distribuído e balanceamento de carga;

2. Particionamento do Serviço Web para permitir a utilização de partes mais fundamentais do sistema **INSANE**, como os recursos de álgebra matricial, geometria computacional, estruturas de dados, persistência e validação de dados, algoritmos para processos incrementais-iterativos de solução de sistemas de equações não-lineares, entre outros;

3. Aprimoramento da aplicação Web através da adoção de tecnologias *RIA* com o objetivo de adicionar recursos gráficos interativos de aplicações Desktop tradicionais ao ***InsaneWeb***;

4. Desenvolvimento de um aplicativo Web para pré-processamento;

5. Desenvolvimento de um cliente para dispositivos microeletrônicos (*palm-tops*, celulares, entre outros);

6. Integração da geração de relatórios no formato texto ao aplicativo **INSANE** Desktop, utilizando as mesmas tecnologias de processamento *XML* e folhas de estilo para visualização em navegadores Web.

Apêndice A

Generalização do Modelo

A generalização realizada no modelo do sistema **INSANE** contribui para que o sistema contemple diversos tipos de problemas que podem ser descritos através do seguinte sistema de equações algébricas:

$$[A]\{\ddot{X}\} + [B]\{\dot{X}\} + [C]\{X\} = \{R\} \tag{A.1}$$

onde {X} é a variável de estado do problema e cada termo depende do significado do mesmo.

A seguir são listados alguns exemplos de problemas de computação científica que podem ser modelados com a generalização aqui apresentada.

1. Problemas estáticos da mecânica dos sólidos:

Termo na equação genérica	**Termo particularizado**	**Descrição**
$[A]$	nulo	é nulo neste tipo de problema
$[B]$	nulo	é nulo neste tipo de problema
$[C]$	$[K]$	matriz de rigidez
$\{X\}$	$\{d\}$	deslocamentos
$\{R\}$	$\{F\}$	força

Equação particularizada:

$$[K]\{d\} = \{F\} \tag{A.2}$$

2. Problemas dinâmicos da mecânica dos sólidos:

Termo na equação genérica	**Termo particularizado**	**Descrição**
$[A]$	$[M]$	massa
$\{\ddot{X}\}$	$\{\ddot{\delta}\}$	aceleração
$[B]$	$[C]$	amortecimento
$\{\dot{X}\}$	$\{\dot{\delta}\}$	velocidades
$[C]$	$[K]$	matriz de rigidez
$\{X\}$	$\{\delta\}$	deslocamentos
$\{R\}$	$\{F(t)\}$	força

Equação particularizada:

$$[M]\{\ddot{\delta}\} + [C]\{\dot{\delta}\} + [K]\{\delta\} = \{F(t)\} \tag{A.3}$$

3. Problemas de campo generalizados:

Termo na equação genérica	**Termo particularizado**	**Descrição**
$[A]$	nulo	é nulo neste tipo de problema
$[B]$	nulo	é nulo neste tipo de problema
$[C]$	$[K] + [K_{s_2}]$	K: depende do problema (independe de ϕ) K_{s_2}: associado a problemas de transferência de calor convectiva
$\{X\}$	$\{\phi\}$	variável de campo
$\{R\}$	$\{R\}$	variável de campo geral

Equação particularizada:

$$[[K] + [K_{s_2}]]\{\phi\} = \{R\} \tag{A.4}$$

4. Problemas de transferência de calor:

Termo na equação genérica	**Termo particularizado**	**Descrição**
$[A]$	nulo	é nulo neste tipo de problema
$[B]$	$[C]$	matriz de capacitância
$\{\dot{X}\}$	$\{\dot{T}\}$	variação de temperatura
$[C]$	$[K_c] + [K_h]$	K_c: matriz de condutância relativa à condução K_h: matriz de condutância relativa à convecção
$\{X\}$	$\{T\}$	distribuição de temperatura
$\{R\}$	$\{\sum R_i\}$	somatório dos vetores de carregamento de calor

Equação particularizada:

$$[C]\{\dot{T}\} + [[K_c] + [K_h]]\{T\} = \{\sum R_i\} \tag{A.5}$$

5. Problemas de mecânica dos fluidos:

Termo na equação genérica	**Termo particularizado**	**Descrição**
$[A]$	nulo	é nulo neste tipo de problema
$[B]$	nulo	é nulo neste tipo de problema
$[C]$	$[K]$	matriz de coeficientes
$\{X\}$	$\{\Phi\}$	velocidade potencial
$\{R\}$	$\{R_1 + R_2\}$	R_1: reações nos nós R_2: valores conhecidos

Equação particularizada:

$$[K]\{\Phi\} = \{R_1 + R_2\} \tag{A.6}$$

Apêndice B

O Núcleo Numérico do INSANE

O núcleo numérico do **INSANE** é hoje (Agosto de 2007), composto de um conjunto de classes e interfaces, sendo que as principais estão representadas no diagrama de classes da Figura B.1.

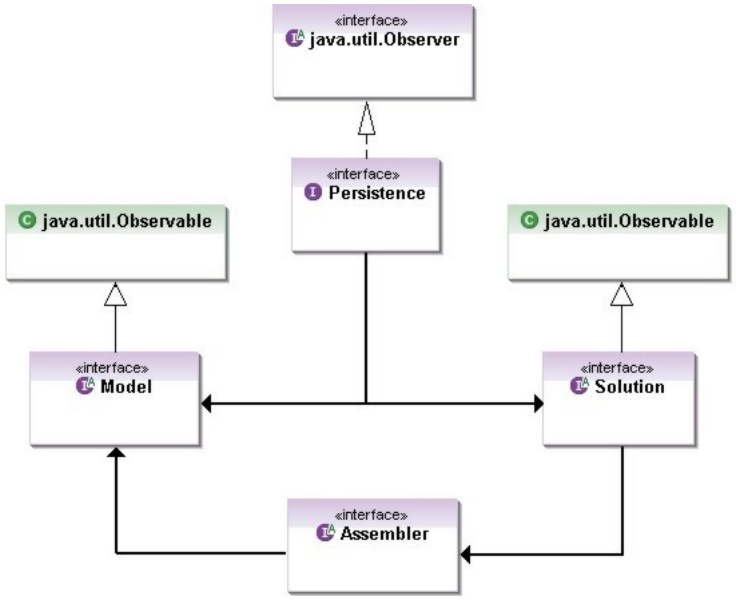


Figura B.1: Organização do núcleo numérico do **INSANE** (Fonseca e Pitangueira, 2007).

A Interface ***Persistence*** padroniza a composição dos dados dos modelos através de arquivos textos (*XML*) ou binários (ISN). A Figura B.2 ilustra esta Interface e as suas três classes já implementadas.

O armazenamento dos dados do modelo em um arquivo *XML* se mostra muito apropriado para o conceito de Serviços Web, que, conforme discutido anteriormente, é baseado no padrão *XML*. O Apêndice E apresenta, como exemplo, um arquivo *XML* de dados **INSANE** e o Apêndice

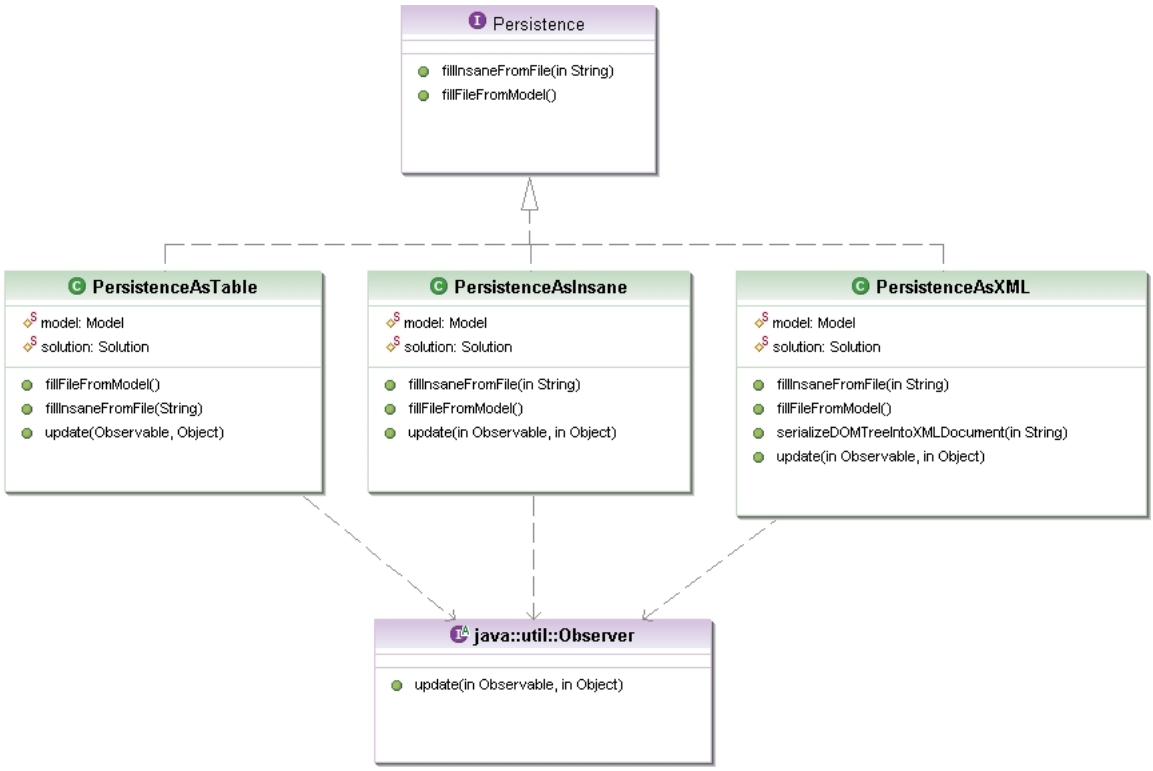


Figura B.2: Interface ***Persistence*** do **INSANE** (Fonseca e Pitangueira, 2007).

D apresenta o modelo de documento, *XML Schema* **INSANE**, que os arquivos de dados devem seguir.

A Interface ***Model*** representa os modelos a serem solucionados, possuindo os dados que compõem o problema. Ela pode ser implementada por classes como, por exemplo, um modelo de elementos finitos ou um modelo de elementos de contorno ou qualquer outro (ver Figura B.3). A classe ***FemModel*** implementa esta Interface para representar um modelo de elementos finitos. Suas principais classes estão representadas na Figura B.4.

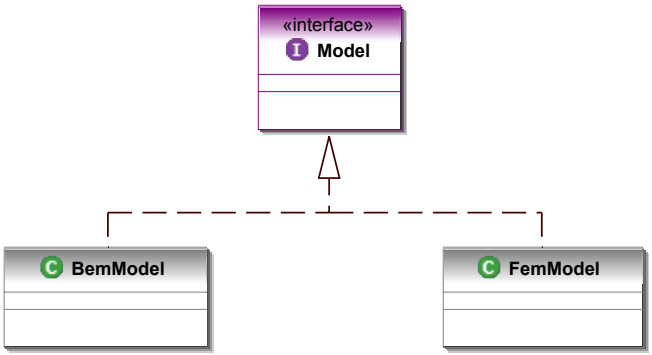


Figura B.3: Interface ***Model*** do **INSANE** (Penna, 2007).

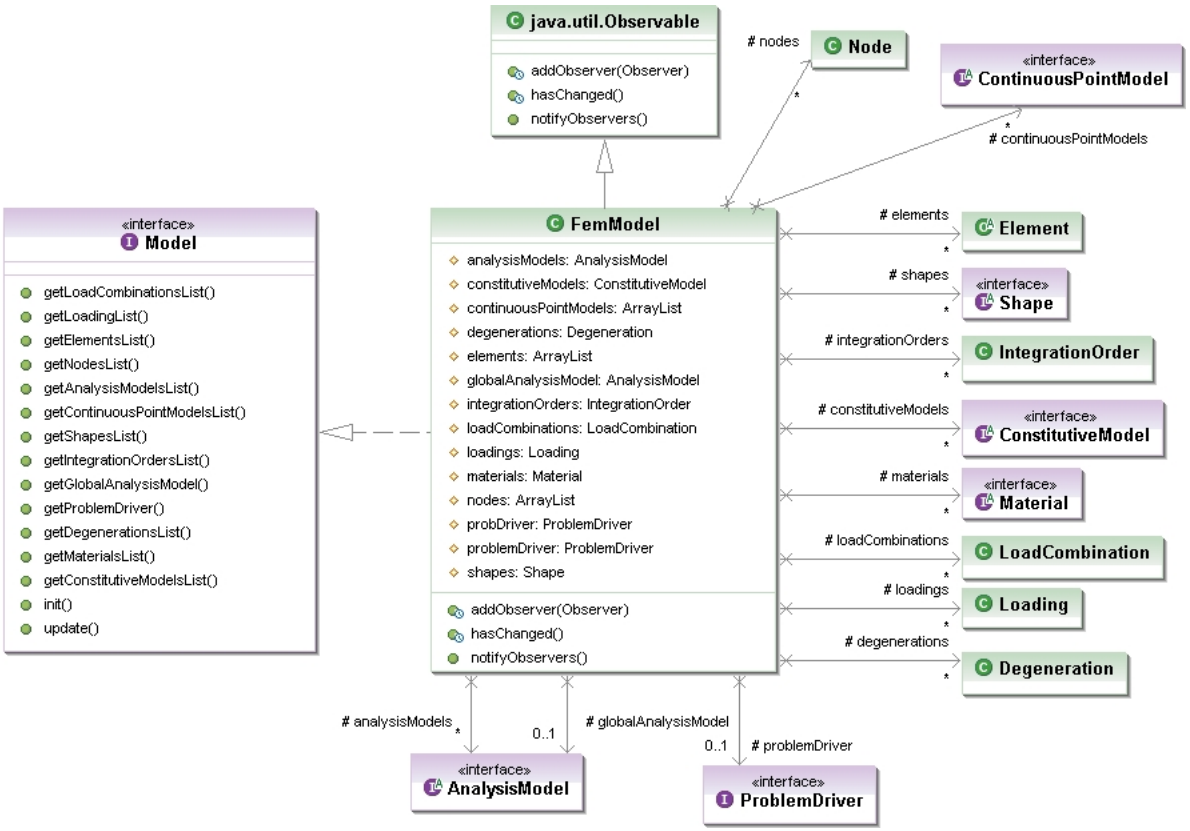


Figura B.4: Classe ***FemModel*** do **INSANE** (Fonseca e Pitangueira, 2007).

Dentre as classes e interfaces que constituem a classe ***FemModel***, vale ressaltar a Interface ***ProblemDriver***, que é a Interface que caracteriza o problema a ser resolvido, de acordo com a equação A.1. É esta Interface a responsável por informar à Interface ***Assembler*** os dados necessários para a montagem da equação de equilíbrio do modelo. A Figura B.5 mostra o diagrama de classes da Interface ***ProblemDriver***. Destacam-se também as classes ***Loading***, que descreve os carregamentos a serem aplicados em cada modelo e ***LoadCombination***, que dá a possibilidade de realizar combinações entre os carregamentos.

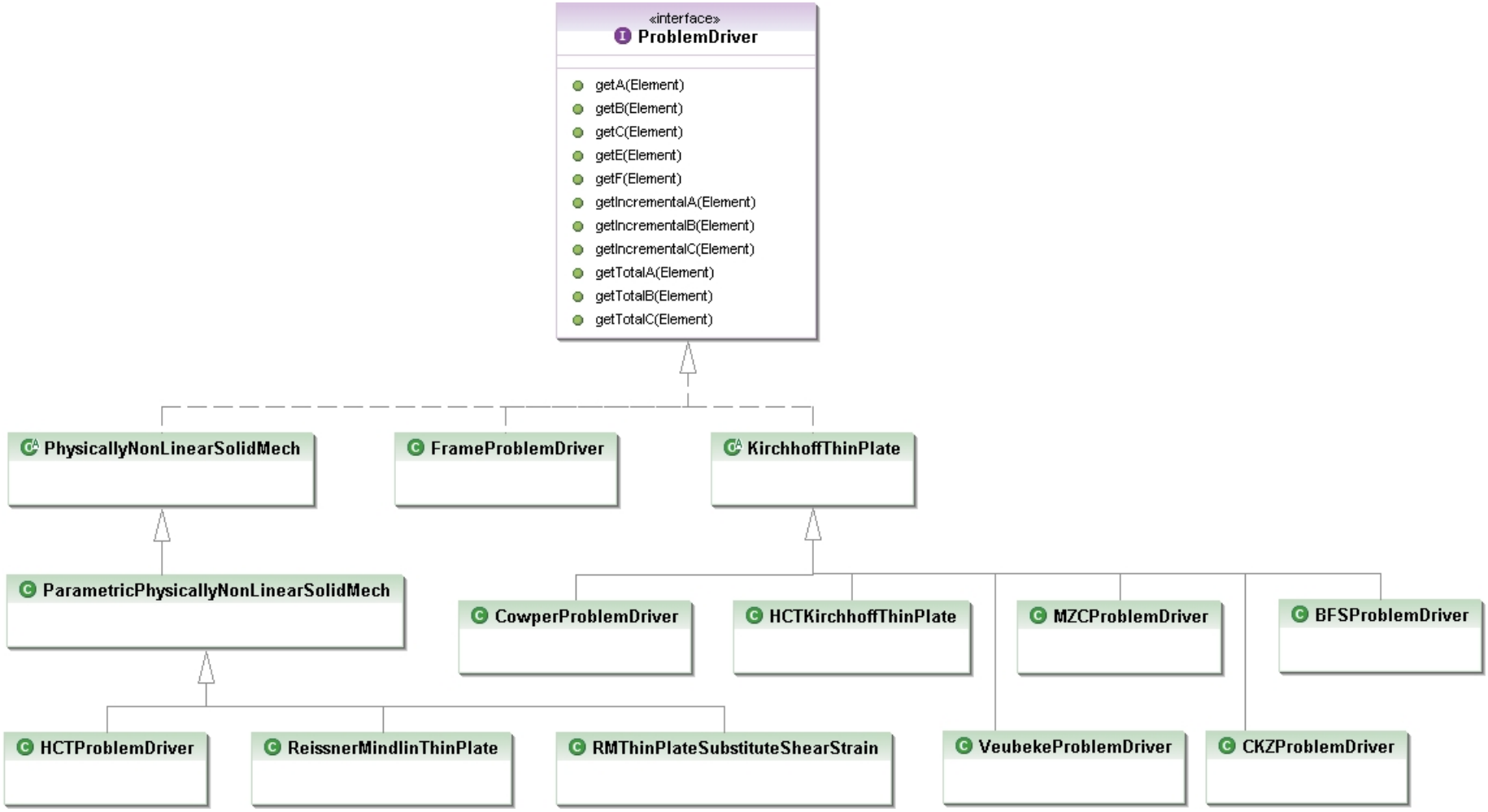


Figura B.5: Diagramas de classes da Interface ***ProblemDriver*** do **INSANE** (Fonseca e Pitangueira, 2007).

A Interface ***Assembler*** monta o sistema de equações (equação A.1) do problema buscando no modelo os dados necessários.

A Interface ***Solution*** (Figura B.6) define os métodos para solucionar a equação do problema. Ela pode caracterizar vários tipos de solução como, por exemplo, a obtenção de uma trajetória de equilíbrio (***EquilibriumPath***) para problemas não-lineares.

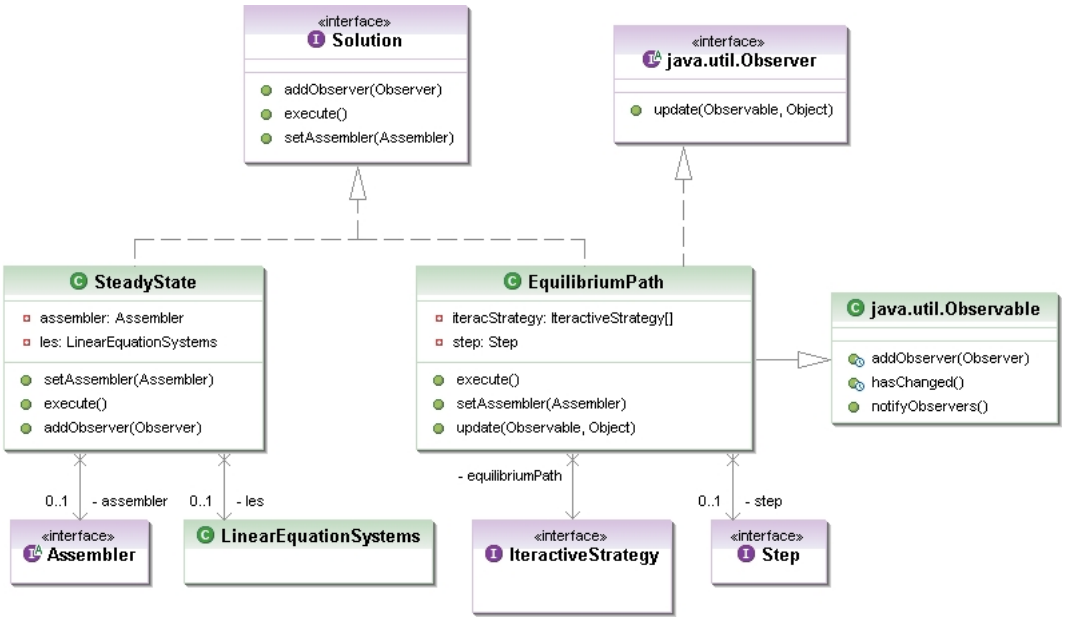


Figura B.6: Interface ***Solution*** do **INSANE** (Fonseca e Pitangueira, 2007).

A Interface ***Shape*** define as classes das funções de forma dos elementos usados na modelagem, como mostra a Figura B.7. Estes elementos são definidos pela classe abstrata ***Element*** cuja hierarquia (Figura B.8) contempla diferentes tipos de elementos.

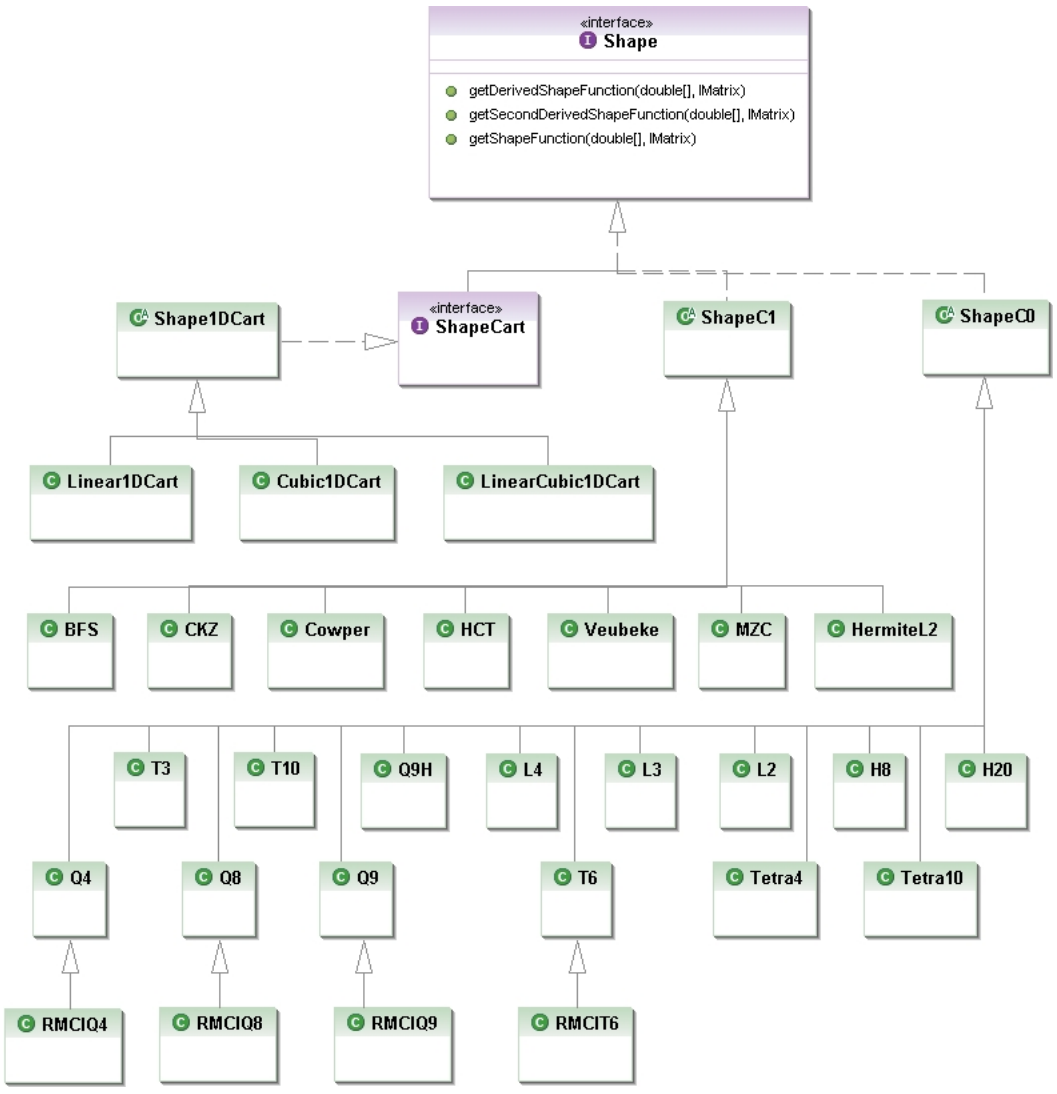


Figura B.7: Interface ***Shape*** do **INSANE** (Fonseca e Pitangueira, 2007).

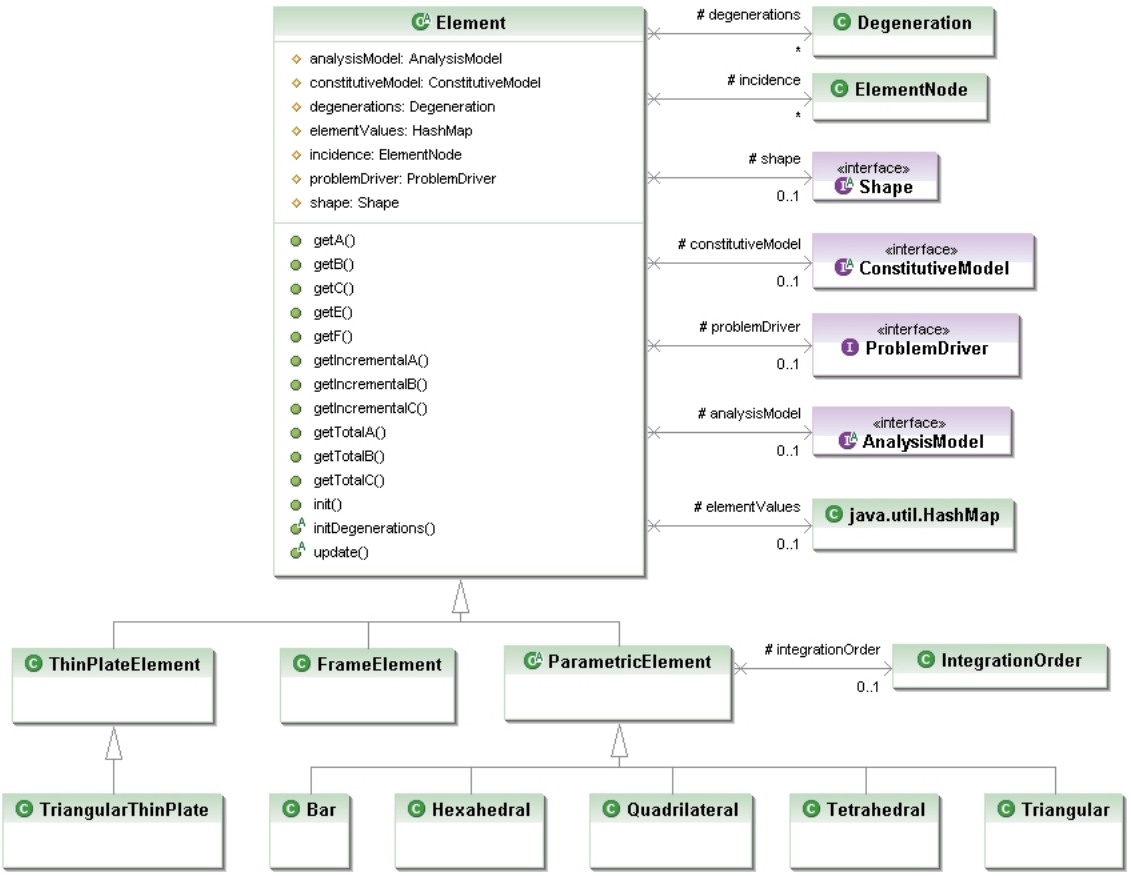


Figura B.8: Classe ***Element*** do **INSANE** (Fonseca e Pitangueira, 2007).

Apêndice C

Tecnologias de Desenvolvimento de Software

A linguagem de programação adotada no desenvolvimento do **INSANE** é a linguagem Java (Sun, 2007). Para o desenvolvimento deste trabalho, a versão utilizada foi Java $1.6.0 - 01$.

O IDE (*Integrated Development Environment*) ***Eclipse*** (`http://www.eclipse.org/`), ferramenta de desenvolvimento Java de código livre mais utilizada no mundo, foi utilizado para realizar as codificações das classes Java do projeto. Ele oferece editor de código-fonte, gerador de código, compilador, depurador e montador, dentre outros inúmeros recursos, como os diversos *plug-in*s de código livre disponíveis gratuitamente na Internet, os quais realizam várias funções adicionais como controle de versões e diagramas UML. A versão do *Eclipse* utilizada neste trabalho foi a 3.2.1.

Além do *Eclipse*, foi utilizado o ***Apache Maven*** (Maven, 2007), para automação e gerenciamento dos vários módulos do sistema **INSANE**. Ele é similar à ferramenta *Ant* (`http://ant.apache.org/`), mas possui um modelo de configuração mais simples, baseado no formato *XML*. O *Maven* utiliza uma construção conhecida como *Project Object Model* para descrever o projeto de software em construção, suas dependências de outros módulos e componentes e a sua seqüencia de construção. Ele oferece tarefas pré-definidas que realizam funções corriqueiras, como compilação, empacotamento de código e execução de testes unitários, automatizando o processo de montagem dos vários módulos em um sistema único e uniforme. É possível, também, programar outros tipos de tarefas, como a execução de *plug-in*s de terceiros. Neste trabalho foi utilizado o *plug-in* do *Axis*2 "`axis2-aar-maven-plugin`" para

geração do arquivo de ***deploy*** do Serviço Web. A versão do *Maven* adotada à época de desenvolvimento, foi: Maven2 versão 2.0.5.

Para o controle de versões do sistema utilizou-se o ***Subversion*** (`http://subversion.tigris.org/`, que é uma ferramenta de apoio ao desenvolvimento colaborativo de software cuja principal função é identificar e controlar sistematicamente as modificações realizadas nos arquivos de um projeto ao longo do tempo. Através de um mecanismo automatizado, ele gerencia a evolução das mudanças e garante a integridade e rastreabilidade das modificações do sistema. O uso de ferramentas para controle de versões é imprescindível em sistemas com vários desenvolvedores trabalhando simultaneamente e localizados em diferentes pontos geográficos. Cada integrante do **INSANE** utiliza um ***plug-in*** do *Subversion* instalado em seu IDE *Eclipse*. Na época do desenvolvimento deste trabalho, foi utilizado o ***plug-in*** *Subclipse* versão 1.2.0 (`http://subclipse.tigris.org/`).

Como ferramenta de visualização da comunicação e relações entre os objetos das aplicações foi adotada a proposta da *Unified Modelling Language* (**UML**), linguagem padronizada para a modelagem de sistemas de software orientados a objetos. Dentre as diversas linguagens gráficas disponíveis, ela é a mais sistematicamente elaborada, e também a mais aceita. As representações gráficas das hierarquias de classes deste trabalho foram confeccionadas com o auxílio do ***plug-in*** para o IDE Eclipse ***EclipseUML Free Edition*** (`http://www.omondo.com/`), segundo os padrões de diagramas UML.

Apêndice D

XML SCHEMA do INSANE

Os Códigos D.1 a D.6 mostram o arquivo "`insane.xsd`" que contém o *XML Schema* do **ISNANE** adotado neste trabalho (Agosto de 2007).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.dees.ufmg.br" xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" targetNamespace="http://www.dees.ufmg.br">

    <xs:element name="Insane">
        <xs:complexType>
            <xs:all minOccurs="1">
                <xs:element ref="Solution"/>
                <xs:element ref="Model"/>
                <xs:element ref="LoadingList"/>
                <xs:element ref="LoadCombinations"/>
            </xs:all>
        </xs:complexType>
    </xs:element>

<!-- XML Schema for SOLUTION -->

        <xs:element name="Solution">
                <xs:complexType>
                    <xs:all>
                        <xs:element minOccurs="0" ref="NumMaxSteps"/>
                        <xs:element minOccurs="0" ref="Step"/>
                        <xs:element minOccurs="0" ref="IterativeStrategyList"/>
                        <xs:element minOccurs="0" ref="ChangeLoadFactor"/>
                        <xs:element minOccurs="0" ref="ChangeTolerance"/>
                        <xs:element minOccurs="0" ref="ChangeNumMaxIterations"/>
                        <xs:element minOccurs="0" ref="StepNumber"/>
                        <xs:element minOccurs="0" ref="FinalLoadFactor"/>
                    </xs:all>
                    <xs:attribute name="class" type="xs:NCName" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="NumMaxSteps" type="xs:integer"/>
            <xs:element name="Step">
                <xs:complexType>
                    <xs:all>
                        <xs:element ref="NumMaxIterations"/>
                        <xs:element ref="Tolerance"/>
                        <xs:element ref="ConvergenceType"/>
                    </xs:all>
```

Código D.1: *XML Schema* do **INSANE** - Agosto de 2007 - 1a. parte.

```
                    <xs:attribute name="class" type="xs:NCName" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="NumMaxIterations" type="xs:integer"/>
            <xs:element name="Tolerance" type="xs:double"/>
            <xs:element name="ConvergenceType" type="xs:integer"/>
            <xs:element name="IterativeStrategyList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element maxOccurs="unbounded" ref="IterativeStrategy"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="IterativeStrategy">
                <xs:complexType>
                    <xs:all>
                        <xs:element minOccurs="0" ref="NodeControl"/>
                        <xs:element minOccurs="0" ref="DirectionControl"/>
                    </xs:all>
                    <xs:attribute name="LoadFactor" type="xs:decimal" use="required"/>
                    <xs:attribute name="class" type="xs:NCName" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="NodeControl" type="xs:string"/>
            <xs:element name="DirectionControl" type="xs:string"/>
            <xs:element name="ChangeLoadFactor" type="xs:double"/>
            <xs:element name="ChangeTolerance" type="xs:double"/>
            <xs:element name="ChangeNumMaxIterations" type="xs:integer"/>
                <xs:element name="StepNumber" type="xs:integer"/>
                <xs:element name="FinalLoadFactor" type="xs:double"/>

<!-- XML Schema for MODEL -->

    <xs:element name="Model">
        <xs:complexType>
            <xs:all>
                <xs:element ref="ProblemDriver"/>
                <xs:element ref="GlobalAnalysisModel"/>
                <xs:element ref="MaterialList"/>
                <xs:element ref="DegenerationList"/>
                <xs:element ref="NodeList"/>
                <xs:element ref="ElementList"/>
            </xs:all>
            <xs:attribute name="class" type="xs:NCName" use="required"/>
        </xs:complexType>
    </xs:element>

<!-- XML Schema for PROBLEM_DRIVER -->

    <xs:element name="ProblemDriver" type="xs:string"/>

<!-- XML Schema for GLOBAL_ANALYSIS_MODEL -->

    <xs:element name="GlobalAnalysisModel" type="xs:string"/>

<!-- XML Schema for MATERIAL_LIST -->
    <xs:element name="MaterialList">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="Material"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Material">
        <xs:complexType>
            <xs:all>
                <xs:element minOccurs="0" ref="Elasticity"/>
                <xs:element minOccurs="0" ref="Elasticity2"/>
                <xs:element minOccurs="0" ref="Elasticity11"/>
                <xs:element minOccurs="0" ref="Elasticity22"/>
                <xs:element minOccurs="0" ref="Elasticity33"/>
```

Código D.2: *XML Schema* do **INSANE** - Agosto de 2007 - 2a. parte.

```
                <xs:element minOccurs="0" ref="Poisson"/>
                <xs:element minOccurs="0" ref="Poisson12"/>
                <xs:element minOccurs="0" ref="Poisson13"/>
                <xs:element minOccurs="0" ref="Poisson23"/>
                <xs:element minOccurs="0" ref="ShearModulus"/>
                <xs:element minOccurs="0" ref="ThermalCoeff"/>
                <xs:element minOccurs="0" ref="Yielding"/>
                <xs:element minOccurs="0" ref="HardeningModulus"/>
                <xs:element minOccurs="0" ref="Fck"/>
                <xs:element minOccurs="0" ref="Fc"/>
                <xs:element minOccurs="0" ref="Ft"/>
                <xs:element minOccurs="0" ref="ec"/>
                <xs:element minOccurs="0" ref="et"/>
                <xs:element minOccurs="0" ref="E0"/>
                <xs:element minOccurs="0" ref="Ec0"/>
                <xs:element minOccurs="0" ref="Ecm"/>
                <xs:element minOccurs="0" ref="Fcm"/>
                <xs:element minOccurs="0" ref="Dc1"/>
                <xs:element minOccurs="0" ref="Dcu"/>
                <xs:element minOccurs="0" ref="LengthBending"/>
                <xs:element minOccurs="0" ref="LengthTorsion"/>
                <xs:element minOccurs="0" ref="Alpha"/>
                <xs:element minOccurs="0" ref="Eta"/>
            </xs:all>
            <xs:attribute name="class" type="xs:NCName" use="required"/>
            <xs:attribute name="label" type="xs:NCName" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Elasticity" type="xs:double"/>
<xs:element name="Elasticity2" type="xs:double"/>
    <xs:element name="Elasticity11" type="xs:double"/>
    <xs:element name="Elasticity22" type="xs:double"/>
        <xs:element name="Elasticity33" type="xs:double"/>
        <xs:element name="Poisson" type="xs:double"/>
    <xs:element name="Poisson12" type="xs:double"/>
        <xs:element name="Poisson13" type="xs:double"/>
        <xs:element name="Poisson23" type="xs:double"/>
        <xs:element name="ShearModulus" type="xs:double"/>
    <xs:element name="ThermalCoeff" type="xs:double"/>
        <xs:element name="Yielding" type="xs:double"/>
        <xs:element name="HardeningModulus" type="xs:double"/>
        <xs:element name="Fck" type="xs:double"/>
        <xs:element name="Fc" type="xs:double"/>
        <xs:element name="Ft" type="xs:double"/>
        <xs:element name="ec" type="xs:double"/>
        <xs:element name="et" type="xs:double"/>
        <xs:element name="E0" type="xs:double"/>
        <xs:element name="Ec0" type="xs:double"/>
        <xs:element name="Ecm" type="xs:double"/>
        <xs:element name="Fcm" type="xs:double"/>
        <xs:element name="Dc1" type="xs:double"/>
        <xs:element name="Dcu" type="xs:double"/>
        <xs:element name="LengthBending" type="xs:double"/>
        <xs:element name="Alpha" type="xs:double"/>
   <xs:element name="LengthTorsion" type="xs:double"/>
        <xs:element name="Eta" type="xs:double"/>

<!-- XML Schema for DEGENERATION_LIST -->

        <xs:element name="DegenerationList">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="Degeneration"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
     <xs:element name="Degeneration">
        <xs:complexType>
            <xs:choice  minOccurs="0" maxOccurs="unbounded">
                <xs:element minOccurs="0" ref="Area"/>
                <xs:element minOccurs="0" ref="Ix"/>
```

Código D.3: *XML Schema* do **INSANE** - Agosto de 2007 - 3a. parte.

```
                <xs:element minOccurs="0" ref="Iy"/>
                <xs:element minOccurs="0" ref="Iz"/>
                <xs:element minOccurs="0" ref="Height"/>
                <xs:element minOccurs="0" ref="CSMaterial"/>
                <xs:element minOccurs="0" ref="Thickness"/>
                <xs:element minOccurs="0" ref="FormFactorY"/>
                <xs:element minOccurs="0" ref="FormFactorZ"/>
                <xs:element maxOccurs="unbounded" minOccurs="0" ref="DegenerationPoint"/>
                <xs:element minOccurs="0" ref="GeometricProperties"/>
            </xs:choice>
            <xs:attribute name="class" type="xs:NCName" use="required"/>
            <xs:attribute name="label" type="xs:NCName" use="required"/>
        </xs:complexType>
    </xs:element>
<xs:element name="Area" type="xs:string"/>
    <xs:element name="Ix" type="xs:double"/>
    <xs:element name="Iy" type="xs:double"/>
    <xs:element name="Iz" type="xs:double"/>
    <xs:element name="Height" type="xs:double"/>
    <xs:element name="CSMaterial" type="xs:NCName"/>
    <xs:element name="Thickness" type="xs:double"/>
    <xs:element name="FormFactorY" type="xs:double"/>
    <xs:element name="FormFactorZ" type="xs:double"/>
    <xs:element name="DegenerationPoint">
        <xs:complexType>
            <xs:all>
                <xs:element ref="PointMaterial"/>
                <xs:element ref="PointAnalysisModel"/>
                <xs:element ref="PointConstitutiveModel"/>
                <xs:element ref="Area"/>
            </xs:all>
            <xs:attribute name="SectionCoord" use="required"/>
            <xs:attribute name="label" type="xs:integer" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="PointMaterial" type="xs:NCName"/>
    <xs:element name="PointAnalysisModel" type="xs:NCName"/>
    <xs:element name="PointConstitutiveModel" type="xs:NCName"/>
    <xs:element name="GeometricProperties">
        <xs:complexType>
            <xs:all>
                <xs:element minOccurs="0" ref="SectionArea"/>
                <xs:element minOccurs="0" ref="InertiaX"/>
                <xs:element minOccurs="0" ref="InertiaY"/>
                <xs:element minOccurs="0" ref="InertiaZ"/>
                <xs:element minOccurs="0" ref="SectionThickness"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:element name="SectionArea" type="xs:double"/>
    <xs:element name="InertiaX" type="xs:double"/>
    <xs:element name="InertiaY" type="xs:double"/>
    <xs:element name="InertiaZ" type="xs:double"/>
    <xs:element name="SectionThickness" type="xs:double"/>

<!-- XML Schema for NODE_LIST -->

        <xs:element name="NodeList">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="Node"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Node">
        <xs:complexType>
            <xs:all>
                <xs:element ref="Coord"/>
                <xs:element minOccurs="0" ref="NodeValues"/>
            </xs:all>
            <xs:attribute name="label" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
```

Código D.4: *XML Schema* do **INSANE** - Agosto de 2007 - 4a. parte.

```
    <xs:element name="Coord" type="xs:string"/>
    <xs:element name="NodeValues">
        <xs:complexType>
            <xs:all>
                <xs:element minOccurs="0" ref="DOFLabels"/>
                <xs:element minOccurs="0" ref="Restraints"/>
                <xs:element minOccurs="0" ref="PreDisplacements"/>
                                <xs:element minOccurs="0" ref="Coeficients"/>
                                <xs:element minOccurs="0" ref="Displacements"/>
                <xs:element minOccurs="0" ref="Reactions"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:element name="DOFLabels" type="xs:string"/>
    <xs:element name="Restraints" type="xs:string"/>
    <xs:element name="PreDisplacements" type="xs:string"/>
        <xs:element name="Coeficients" type="xs:string"/>
        <xs:element name="Displacements" type="xs:string"/>
    <xs:element name="Reactions" type="xs:string"/>
<!-- XML Schema for ELEMENT_LIST -->

        <xs:element name="ElementList">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="Element"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Element">
        <xs:complexType>
            <xs:all>
                <xs:element ref="Incidence"/>
                <xs:element ref="AnalysisModel"/>
                <xs:element minOccurs="0" ref="IntegrationOrder"/>
                <xs:element minOccurs="0" ref="ConstitutiveModel"/>
                <xs:element ref="ElmDegenerations"/>
                <xs:element minOccurs="0" ref="IntVarLabels"/>
                <xs:element minOccurs="0" ref="DualIntVarLabels"/>
                <xs:element minOccurs="0" ref="ElmDegenerationList"/>
            </xs:all>
            <xs:attribute name="class" use="required"/>
            <xs:attribute name="label" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Incidence" type="xs:string"/>
    <xs:element name="AnalysisModel" type="xs:NCName"/>
    <xs:element name="IntegrationOrder" type="xs:string"/>
    <xs:element name="ConstitutiveModel" type="xs:NCName"/>
    <xs:element name="ElmDegenerations" type="xs:string"/>
    <xs:element name="IntVarLabels" type="xs:string"/>
    <xs:element name="DualIntVarLabels" type="xs:string"/>
    <xs:element name="ElmDegenerationList">
        <xs:complexType>
                <xs:sequence>
                <xs:element maxOccurs="unbounded" minOccurs="0" ref="ElmDegeneration"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ElmDegeneration">
        <xs:complexType>
                <xs:choice  minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="IPCoord"/>
                <xs:element ref="GeneralizedStrains"/>
                <xs:element ref="GeneralizedStresses"/>
                <xs:element ref="MPIntVarLabels"/>
                <xs:element ref="MPDualIntVarLabels"/>
                <xs:element maxOccurs="unbounded" ref="SectionPoint"/>
            </xs:choice>
            <xs:attribute name="label" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
```

Código D.5: *XML Schema* do **INSANE** - Agosto de 2007 - 5a. parte.

```
    <xs:element name="IPCoord" type="xs:string"/>
    <xs:element name="GeneralizedStrains" type="xs:string"/>
    <xs:element name="GeneralizedStresses" type="xs:string"/>
    <xs:element name="MPIntVarLabels" type="xs:string"/>
    <xs:element name="MPDualIntVarLabels" type="xs:string"/>
    <xs:element name="SectionPoint">
        <xs:complexType>
                <xs:all>
                <xs:element ref="Strains"/>
                <xs:element ref="Stresses"/>
            </xs:all>
            <xs:attribute name="label" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Strains" type="xs:string"/>
    <xs:element name="Stresses" type="xs:string"/>

<!-- XML Schema for LOADING_LIST -->

        <xs:element name="LoadingList">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="Loading"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Loading">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" minOccurs="0" ref="NodeLoad"/>
            </xs:sequence>
            <xs:attribute name="label" type="xs:integer" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="NodeLoad">
        <xs:complexType mixed="true">
            <xs:attribute name="node" type="xs:double" use="required"/>
        </xs:complexType>
    </xs:element>

<!-- XML Schema for LOAD_COMBINATION -->

        <xs:element name="LoadCombinations">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="LoadCombination"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="LoadCombination">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="LoadCase"/>
            </xs:sequence>
            <xs:attribute name="label" type="xs:integer" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="LoadCase">
        <xs:complexType>
            <xs:attribute name="factor" type="xs:double" use="required"/>
            <xs:attribute name="inc" type="xs:boolean" use="required"/>
            <xs:attribute name="loading" type="xs:double" use="required"/>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Código D.6: *XML Schema* do **INSANE** - Agosto de 2007 - 6a. parte.

Apêndice E

Exemplo de Arquivo de Dados XML do INSANE

Os Códigos E.1 e E.2 mostram o arquivo de dados **INSANE** "`kelson1_xsd.xml`", em formato *XML*, para um modelo de estado plano de tensões.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Insane xmlns="http://www.dees.ufmg.br" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.dees.ufmg.br insane.xsd">
        <Solution class="SteadyState"/>
        <Model class="FemModel">
                <ProblemDriver>ParametricPhysicallyNonLinearSolidMech</ProblemDriver>
                <GlobalAnalysisModel>PlaneStress</GlobalAnalysisModel>
                <MaterialList>
                        <Material class="LinearElasticIsotropic" label="LinearElasticIsotropic">
                                <Elasticity>2000000</Elasticity>
                                <Poisson>0.2</Poisson>
                                <ShearModulus>1</ShearModulus>
                        </Material>
                </MaterialList>
                <DegenerationList>
                        <Degeneration class ="PrescribedDegeneration" label="dg1">
                                <CSMaterial>LinearElasticIsotropic</CSMaterial>
                                <Thickness>0.01</Thickness>
                        </Degeneration>
                </DegenerationList>
                <NodeList>
                        <Node label="1">
                                <Coord>0.0 4.0 0.0</Coord>
                        </Node>
                        <Node label="2">
                                <Coord>2.0 4.0 0.0</Coord>
                        </Node>
                        <Node label="3">
                                <Coord>4.0 4.0 0.0</Coord>
                        </Node>
                        <Node label="4">
                                <Coord>1.0 3.0 0.0</Coord>
                        </Node>
                        <Node label="5">
                                <Coord>3.0 3.0 0.0</Coord>
                        </Node>
                        <Node label="6">
```

Código E.1: Exemplo de um arquivo *XML* contendo os dados do modelo **INSANE** - 1a. Parte.

```
                        <Coord>0.0 2.0 0.0</Coord>
                    </Node>
                    <Node label="7">
                        <Coord>2.0 2.0 0.0</Coord>
                    </Node>
                    <Node label="8">
                        <Coord>4.0 2.0 0.0</Coord>
                    </Node>
                    <Node label="9">
                        <Coord>1.0 1.0 0.0</Coord>
                    </Node>
                    <Node label="10">
                        <Coord>3.0 1.0 0.0</Coord>
                    </Node>
                    <Node label="11">
                        <Coord>0.0 0.0 0.0</Coord>
                        <NodeValues>
                            <Restraints>true true</Restraints>
                        </NodeValues>
                    </Node>
                    <Node label="12">
                        <Coord>2.0 0.0 0.0</Coord>
                    </Node>
                    <Node label="13">
                        <Coord>4.0 0.0 0.0</Coord>
                        <NodeValues>
                            <Restraints>false true</Restraints>
                        </NodeValues>
                    </Node>
                </NodeList>
                <ElementList>
                    <Element class="ParametricElement.Triangular.T6" label="1">
                        <Incidence>1 4 7 5 3 2</Incidence>
                        <AnalysisModel>PlaneStress</AnalysisModel>
                        <IntegrationOrder>3 0 0</IntegrationOrder>
                        <ConstitutiveModel>LinearElasticConstModel</ConstitutiveModel>
                        <ElmDegenerations>dg1 dg1 dg1</ElmDegenerations>
                    </Element>
                    <Element class="ParametricElement.Triangular.T6" label="2">
                        <Incidence>11 9 7 4 1 6</Incidence>
                        <AnalysisModel>PlaneStress</AnalysisModel>
                        <IntegrationOrder>3 0 0</IntegrationOrder>
                        <ConstitutiveModel>LinearElasticConstModel</ConstitutiveModel>
                        <ElmDegenerations>dg1 dg1 dg1</ElmDegenerations>
                    </Element>
                    <Element class="ParametricElement.Triangular.T6" label="3">
                        <Incidence>7 10 13 8 3 5</Incidence>
                        <AnalysisModel>PlaneStress</AnalysisModel>
                        <IntegrationOrder>3 0 0</IntegrationOrder>
                        <ConstitutiveModel>LinearElasticConstModel</ConstitutiveModel>
                        <ElmDegenerations>dg1 dg1 dg1</ElmDegenerations>
                    </Element>
                    <Element class="ParametricElement.Triangular.T6" label="4">
                        <Incidence>11 12 13 10 7 9</Incidence>
                        <AnalysisModel>PlaneStress</AnalysisModel>
                        <IntegrationOrder>3 0 0</IntegrationOrder>
                        <ConstitutiveModel>LinearElasticConstModel</ConstitutiveModel>
                        <ElmDegenerations>dg1 dg1 dg1</ElmDegenerations>
                    </Element>
                </ElementList>
        </Model>
        <LoadingList>
                <Loading label="1">
                    <NodeLoad node="2">0 -100.0</NodeLoad>
                </Loading>
        </LoadingList>
        <LoadCombinations>
                <LoadCombination label="1">
                    <LoadCase loading="1" inc="false" factor="1.0"/>
                </LoadCombination>
        </LoadCombinations>
</Insane>
```

Código E.2: Exemplo de um arquivo *XML* contendo os dados do modelo **INSANE** - 2a. Parte.

Apêndice F

WSDL do Serviço Web do INSANE

Os Códigos F.1 a F.4 mostram o arquivo "`InsaneService.wsdl`" que contém o descritor *WSDL* para o Serviço Web **INSANE** (também disponível em: `http://insane.dees.ufmg.br:8080/axis2/services/InsaneService`).

```
<wsdl:definitions xmlns:ns="http://webService.insane.dees.ufmg.br"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:xsd="http://webService.insane.dees.ufmg.br/xsd"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
 targetNamespace="http://webService.insane.dees.ufmg.br">
 <wsdl:types>
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="qualified" elementFormDefault="qualified"
    targetNamespace="http://webService.insane.dees.ufmg.br/xsd">
    <xs:element name="getModelKeys">
        <xs:complexType>
          <xs:sequence>
                <xs:element name="param0" nillable="true"
                 type="xs:anyType" />
          </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getModelKeysResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" name="return"
                nillable="true" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getFigure">
        <xs:complexType>
          <xs:sequence>
                <xs:element name="param0" nillable="true"
                 type="xs:anyType" />
          </xs:sequence>
        </xs:complexType>
    </xs:element>
```

Código F.1: `InsaneService.wsdl` - 1a. Parte.

```
  <xs:element name="getFigureResponse">
     <xs:complexType>
       <xs:sequence>
            <xs:element name="return" nillable="true"
                type="xs:anyType" />
        </xs:sequence>
     </xs:complexType>
   </xs:element>
   <xs:element name="getModelSolved">
        <xs:complexType>
          <xs:sequence>
                <xs:element name="param0" nillable="true"
                    type="xs:anyType" />
                <xs:element name="param1" nillable="true"
                    type="xs:string" />
          </xs:sequence>
        </xs:complexType>
   </xs:element>
   <xs:element name="getModelSolvedResponse">
        <xs:complexType>
           <xs:sequence>
                <xs:element name="return" nillable="true"
                        type="xs:anyType" />
           </xs:sequence>
        </xs:complexType>
   </xs:element>
   <xs:element name="getResultFigure">
        <xs:complexType>
           <xs:sequence>
                <xs:element name="param0" nillable="true"
                        type="xs:anyType" />
                <xs:element name="param1" nillable="true"
                        type="xs:string" />
           </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getResultFigureResponse">
        <xs:complexType>
           <xs:sequence>
                <xs:element name="return" nillable="true"
                        type="xs:anyType" />
           </xs:sequence>
        </xs:complexType>
    </xs:element>
  </xs:schema>
</wsdl:types>
<wsdl:message name="getModelKeysMessage">
  <wsdl:part name="part1" element="xsd:getModelKeys" />
</wsdl:message>
<wsdl:message name="getModelKeysResponseMessage">
  <wsdl:part name="part1" element="xsd:getModelKeysResponse" />
</wsdl:message>
<wsdl:message name="getFigureMessage">
  <wsdl:part name="part1" element="xsd:getFigure" />
</wsdl:message>
<wsdl:message name="getFigureResponseMessage">
  <wsdl:part name="part1" element="xsd:getFigureResponse" />
</wsdl:message>
<wsdl:message name="getModelSolvedMessage">
  <wsdl:part name="part1" element="xsd:getModelSolved" />
</wsdl:message>
<wsdl:message name="getModelSolvedResponseMessage">
   <wsdl:part name="part1" element="xsd:getModelSolvedResponse" />
</wsdl:message>
<wsdl:message name="getResultFigureMessage">
   <wsdl:part name="part1" element="xsd:getResultFigure" />
</wsdl:message>
<wsdl:message name="getResultFigureResponseMessage">
   <wsdl:part name="part1" element="xsd:getResultFigureResponse" />
</wsdl:message>
```

Código F.2: `InsaneService.wsdl` - 2a. Parte.

```
 <wsdl:portType name="InsaneServicePortType">
  <wsdl:operation name="getModelKeys">
        <wsdl:input message="ns:getModelKeysMessage" />
        <wsdl:output message="ns:getModelKeysResponseMessage" />
  </wsdl:operation>
  <wsdl:operation name="getFigure">
        <wsdl:input message="ns:getFigureMessage" />
        <wsdl:output message="ns:getFigureResponseMessage" />
  </wsdl:operation>
  <wsdl:operation name="getModelSolved">
        <wsdl:input message="ns:getModelSolvedMessage" />
        <wsdl:output message="ns:getModelSolvedResponseMessage" />
  </wsdl:operation>
  <wsdl:operation name="getResultFigure">
        <wsdl:input message="ns:getResultFigureMessage" />
        <wsdl:output message="ns:getResultFigureResponseMessage" />
  </wsdl:operation>
 </wsdl:portType>
<wsdl:binding name="InsaneServiceSOAP11Binding"
        type="ns:InsaneServicePortType">
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
 <wsdl:operation name="getModelKeys">
        <soap:operation soapAction="urn:getModelKeys"
              style="document" />
        <wsdl:input>
          <soap:body use="literal"
                namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal"
                namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getFigure">
        <soap:operation soapAction="urn:getFigure" style="document" />
        <wsdl:input>
          <soap:body use="literal"
                namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal"
                namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getModelSolved">
        <soap:operation soapAction="urn:getModelSolved"
                style="document" />
        <wsdl:input>
          <soap:body use="literal"
                namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal"
                namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getResultFigure">
        <soap:operation soapAction="urn:getResultFigure"
                style="document" />
        <wsdl:input>
          <soap:body use="literal"
                namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal"
                namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:output>
```

Código F.3: `InsaneService.wsdl` - 3a. Parte.

```
   </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="InsaneServiceSOAP12Binding"
        type="ns:InsaneServicePortType">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
  <wsdl:operation name="getModelKeys">
        <soap12:operation soapAction="urn:getModelKeys"
                style="document" />
        <wsdl:input>
                <soap12:body use="literal"
                        namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:input>
        <wsdl:output>
                <soap12:body use="literal"
                        namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getFigure">
        <soap12:operation soapAction="urn:getFigure"
                style="document" />
        <wsdl:input>
                <soap12:body use="literal"
                        namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:input>
        <wsdl:output>
                <soap12:body use="literal"
                        namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getModelSolved">
        <soap12:operation soapAction="urn:getModelSolved"
                style="document" />
        <wsdl:input>
                <soap12:body use="literal"
                        namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:input>
        <wsdl:output>
                <soap12:body use="literal"
                        namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getResultFigure">
        <soap12:operation soapAction="urn:getResultFigure"
                style="document" />
        <wsdl:input>
                <soap12:body use="literal"
                        namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:input>
        <wsdl:output>
                <soap12:body use="literal"
                        namespace="http://webService.insane.dees.ufmg.br" />
        </wsdl:output>
   </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="InsaneService">
   <wsdl:port name="InsaneServiceSOAP11port"
        binding="ns:InsaneServiceSOAP11Binding">
      <soap:address
         location="http://localhost:8080/axis2/services/InsaneService"/>
   </wsdl:port>
   <wsdl:port name="InsaneServiceSOAP12port"
          binding="ns:InsaneServiceSOAP12Binding">
        <soap12:address
          location="http://localhost:8080/axis2/services/InsaneService"/>
   </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Código F.4: `InsaneService.wsdl` - 4a. Parte.

Apêndice G

Detalhamento das Tecnologias Relacionadas a Serviços Web

G.1 Introdução

Neste apêndice são apresentados maiores detalhes sobre as tecnologias relacionadas a Serviços Web.

G.2 Hypertext Transfer Protocol (HTTP)

O *HTTP* é o protocolo que permite aos Serviços Web e navegadores enviar e receber dados pela Internet (Kurniawan e Deck, 2004). Clientes e servidores se comunicam através de mensagens *HTTP* de requisição (*Request*) e resposta (*Response*), utilizando conexões TCP.

Uma requisição *HTTP* é formada por três partes (ver Figura G.1):

(i) Método + *URI*[1] + Protocolo/Versão;

(ii) Cabeçalhos;

(iii) Corpo da requisição;

A primeira parte da requisição *HTTP* é composta por uma linha que contém a definição do método, a *URI* e o protocolo da mensagem.

[1] *URI* ou *Uniform Resource Identifier* é uma cadeia de caracteres que identifica um recurso na Web.

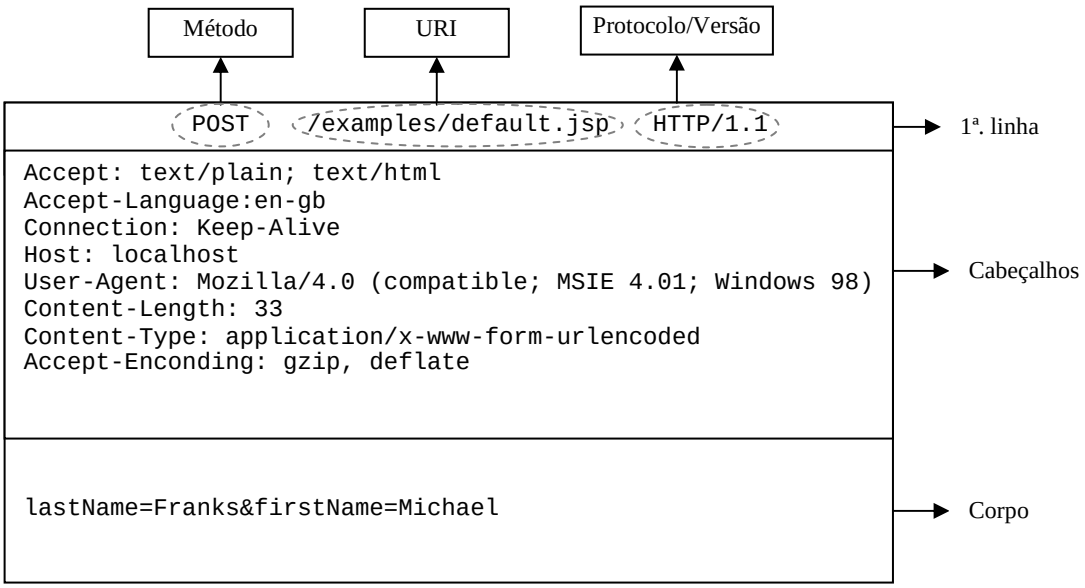


Figura G.1: Exemplo de uma requisição *HTTP* (Kurniawan e Deck, 2004).

O padrão *HTTP* permite sete diferentes métodos para as suas requisições, que são: GET, POST, HEAD, OPTIONS, PUT, DELETE e TRACE. Os métodos GET e POST são os mais usados em aplicações via Internet.

O método GET é usado em requisições cuja finalidade é obter arquivos estáticos, como um documento *HTML* ou um arquivo de imagem.

O método POST é usado para enviar informações ao servidor, as quais são dispostas no corpo da requisição.

O *URI* faz a especificação do recurso de Internet. Normalmente ela é interpretada como sendo o caminho relativo ao diretório raiz do servidor, por isso ela é escrita começando com uma barra.

Finalizando a primeira parte da requisição *HTTP*, vem a descrição da versão do protocolo *HTTP* que está sendo usada.

A segunda parte da requisição *HTTP* é composta pelos cabeçalhos que contém informações úteis sobre o ambiente do lado do cliente e sobre o corpo da mensagem. Eles podem informar, por exemplo, o idioma usado no navegador, o tamanho da mensagem, etc.

Entre os cabeçalhos e o corpo da mensagem vem uma linha em branco, indicando o fim da segunda parte da requisição.

A terceira parte é formada pelo corpo da mensagem que é o conteúdo a ser transmitido pela requisição *HTTP*.

Uma resposta *HTTP* é formada, também, por três partes (ver Figura G.2):

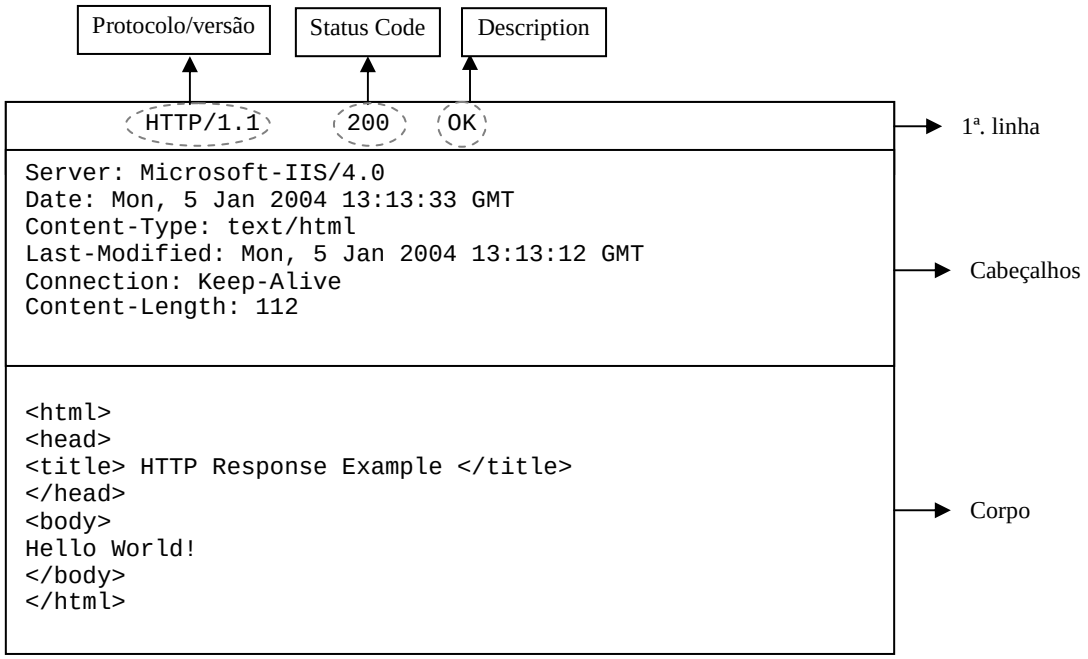


Figura G.2: Exemplo de uma resposta *HTTP* (Kurniawan e Deck, 2004).

(i) Protocolo/Versão + Código de "Status" + Descrição;

(ii) Cabeçalhos;

(iii) Corpo da resposta;

O formato de uma resposta *HTTP* é similar ao da requisição. O código de "*Status*" é um número padronizado que representa o sucesso da requisição (200 para bem sucedida, 404 para falha, dentre outros).

G.3 XML

XML significa "*Extensible Markup Language*" ou, em português, Linguagem de Marcação Extensível.

Entretanto, a *XML* não é apenas uma linguagem de marcação, mas um conjunto de regras para a criação de linguagens de marcação. Enquanto a *HTML* é usada para formatação e exibição de informações, a *XML* é usada para descrever e armazenar essas informações.

Assim como a *HTML*, a *XML* usa marcas e atributos para formatação de elementos, mas, por ser extensível, a *XML* permite a criação de elementos. As marcas *XML* definem uma estrutura para a informação.

A estrutura de um documento *XML* é composta por:

- Declaração *XML* (obrigatória);
- Declaração do tipo de documento;
- Instruções de processamento;
- Comentários;
- Texto (dados);
- Elementos e seus atributos;
- Definição do tipo de documento (DTD ou *XML Schema*).

G.3.1 PRÓLOGO

A declaração *XML* e a declaração do tipo de documento são normalmente chamadas de "prólogo". A Figura G.3 mostra um prólogo de um documento *XML*.

Conforme mostra a Figura G.3, o prólogo contém três partes:

(1) **Declaração** `XML`**:** identifica o documento como *XML* e informa a versão *XML* usada, no caso, versão 1.0 . Usa-se a marca: `<?xml ... ?>`;

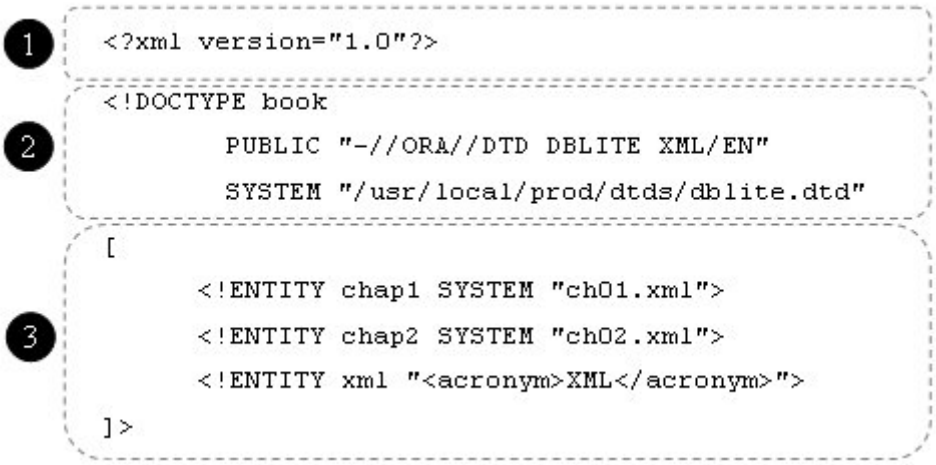


Figura G.3: Exemplo de prólogo de um documento *XML* (Ray, 2001).

(2) **Declaração do tipo de documento, que é composta por:**

- Marca: <!DOCTYPE;
- Elemento raiz (principal): *book*;
- Identificador público: *PUBLIC*, que indica que o documento segue as regras de um DTD disponível publicamente;
- Definição do documento público: "-//ORA//DTD DBLITE XML/EN";
- Identificador de sistema: *SYSTEM*, que especifica o local da DTD;
- Localização física da DTD: "`usr/local/prod/dtds/dblite.dtd`".

(3) **Subconjunto interno:** local para declarações especiais. Neste exemplo, existem três declarações de entidade, identificadas pela marca "<!ENTITY>". As entidades são um recurso especial para substituição de texto em partes do documento. Este subconjunto é delimitado por colchetes [ ]. A marca de fechamento ">" delimita o fim da marca "<!DOCTYPE".

G.3.2 ELEMENTOS

Os elementos dividem o documento em suas partes constituintes. Eles podem conter texto, outros elementos ou ambos.

Os elementos podem ser do tipo contêiner, que delimitam uma região e podem conter texto e outros elementos. A Figura G.4 mostra a sintaxe para um elemento deste tipo.

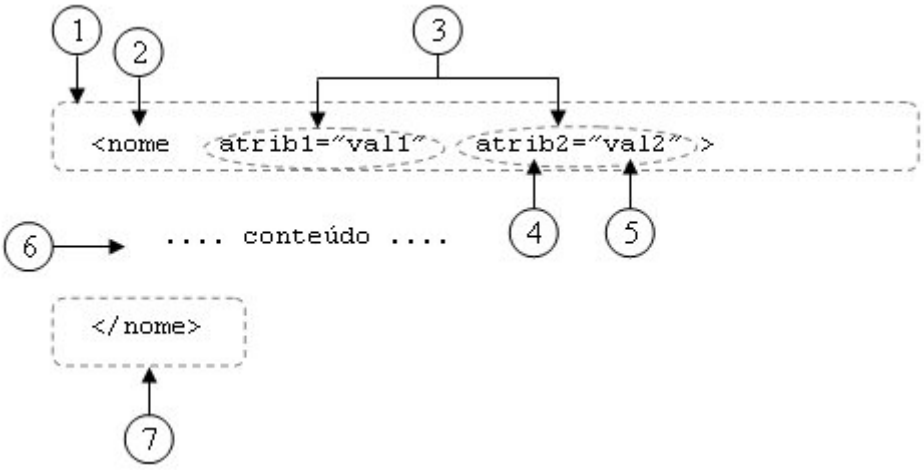


Figura G.4: Sintaxe de um elemento contêiner (Ray, 2001).

A primeira parte do elemento é a marca de início (1), que consiste em um sinal de menor seguido pelo nome da marca (2). O elemento pode conter atributos (3) separados por um espaço em branco, e termina com um sinal de maior. Um atributo define uma propriedade do elemento e é formado por um nome (4) associado por um sinal de igual a um valor entre aspas (5).

Após a marca de início está o conteúdo do elemento (6) e, em seguida, vem a marca de fim (7). A marca de fim é formada pelo sinal de menor, uma barra, o nome do elemento e um sinal de maior.

Os elementos também podem ser do tipo vazios quando não possuem nenhum conteúdo. A Figura G.5 mostra a sintaxe para um elemento deste tipo. Ele é formado por apenas uma marca, com a sintaxe semelhante à marca de início do elemento contêiner, mas terminando com uma barra seguida pelo sinal de maior (6).

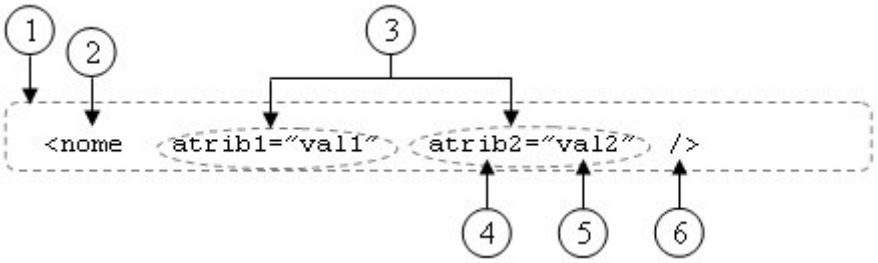


Figura G.5: Sintaxe de um elemento vazio (Ray, 2001).

Um elemento dentro de outro elemento define uma hierarquia facilmente visualizada na codificação *XML*. O elemento raiz ou principal deve ser único e deve conter todos os outros elementos do documento.

O Código G.1 mostra o conteúdo do documento de exemplo "`book.xml`".

```
<?xml version="1.0"?> <!DOCTYPE book
    PUBLIC "-//ORA//DTD DBLITE XML/EN"
    SYSTEM "/usr/local/prod/dtds/dblite.dtd"
[
    <!ENTITY chap1 SYSTEM "ch01.xml">
    <!ENTITY chap2 SYSTEM "ch02.xml">
    <!ENTITY xml "<acronym>XML</acronym>">
]>
<book>
  <title>User Manual</title>
  <author>Indigo Riceway</author>
  <preface id="preface">
     <title>Preface</title>
     <sect1 id="about">
       <title>Availability</title>
<!-- Nota para o autor: talvez colocar uma figura aqui: -->
       <para>
         The information in this manual is available in the following
         forms:
       </para>
       <itemizedlist>
          <listitem> Instant telepathic injection </listitem>
          <listitem> Lumino-google display </listitem>
                ....
          </itemizedlist>
    </sect1>
  </preface>
  <chapter id="intro">
     <title>Introduction</title>
     <para>
            Congratulations on your purchase of one of the most
            ......
     </para>
  </chapter>
</book>
```

Código G.1: `book.xml` - adaptado de Ray (2001).

Neste exemplo, o elemento principal (ou raiz), conforme mencionado na declaração do tipo de documento no prólogo, é "*book*" e ele contém todos os outros elementos, os quais estão descritos a seguir:

- O elemento **title**, cujo conteúdo é *"User Manual"*.
- O elemento **author**, cujo conteúdo é *"Indigo Riceway"*.
- O elemento **preface**, cujo atributo **id** é *"preface"*. Este elemento contém 2 elementos:
  - O elemento **title**, cujo conteúdo é *"Preface"*.
  - O elemento **sect1**, cujo atributo **id** é *"about"*. Este elemento contém 3 elementos:
    - O elemento **title**, cujo conteúdo é *"Availability"*.
    - O elemento **para**, cujo conteúdo é *"The information in this manual is available in the following forms:"*.
    - O elemento **itemizedlist** que contém 2 elementos:
      - O elemento **listitem**, cujo conteúdo é *"Instant telepathic injection"*.
      - O elemento **listitem**, cujo conteúdo é *"Lumino-google display"*.
- O elemento **chapter**, cujo atributo **id** é *"intro"*. Este elemento contém 2 elementos:
  - O elemento **title**, cujo conteúdo é *"Introduction"*.
  - O elemento **para**, cujo conteúdo é *"Congratulations on your purchase of one of the most ....."*.

A hierarquia formada pelo aninhamento dos elementos do Código G.1 pode ser visualizada na Figura G.6.

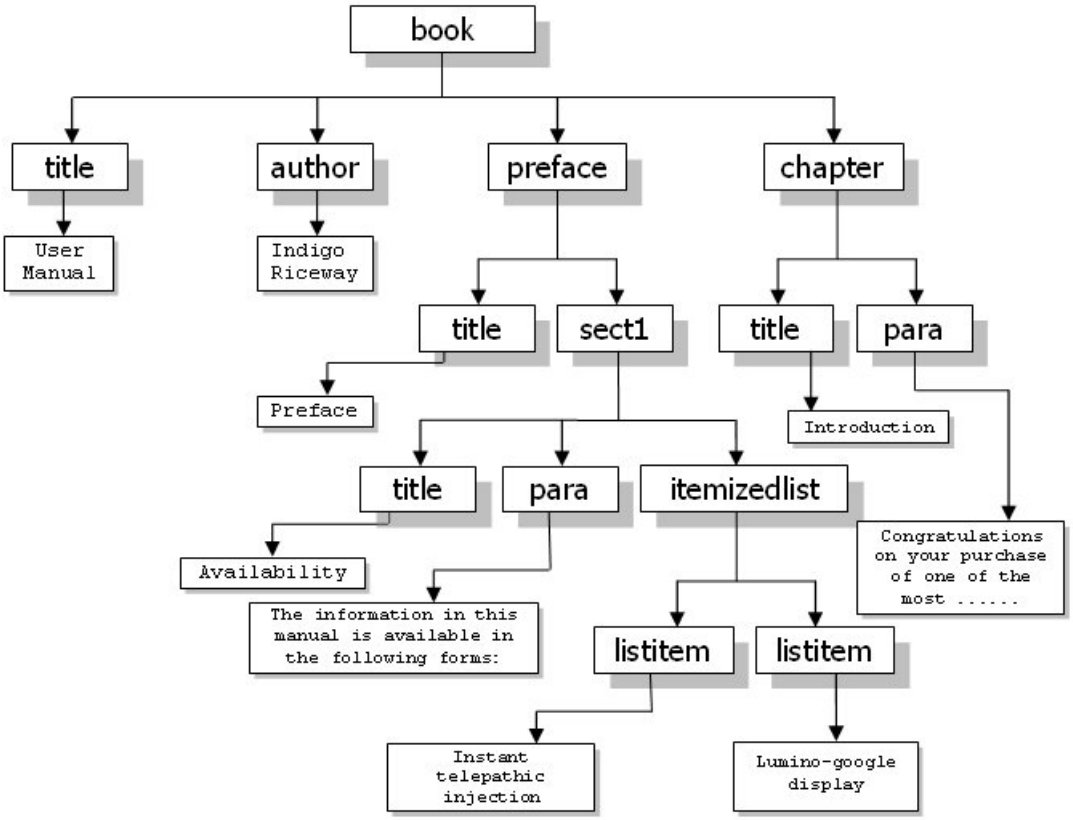


Figura G.6: Hierarquia de elementos *XML* do Código G.1.

G.3.3 COMENTÁRIOS

Os comentários são notas feitas no documento que não são interpretadas pelo analisador *XML*[2]. Assim como nas linguagens de programação de computador, os comentários são úteis para documentar o código e auxiliar o próprio autor ou outros leitores do código.

Um comentário é delimitado no início pelo sinal de menor seguido de um ponto de exclamação e dois traços e, no fim, por dois traços seguidos de um sinal de maior. No Código G.1, é feito o seguinte comentário:

```
<!-- Nota para o autor: talvez colocar uma figura aqui: -->
```

[2]Analisador *XML* é o programa que irá ler e processar o arquivo *XML*.

G.3.4 NAMESPACES

Um documento *XML* pode ser composto, ou seja, pode incorporar documentos *XML* externos para compor um novo documento. Esta composição pode ocasionar dois problemas: reconhecimento e colisão de elementos. O problema do reconhecimento ocorre quando o processador *XML* não consegue identificar quais elementos do novo documento pertencem aos documentos que o originaram. O problema da colisão acontece quando os documentos de origem possuem elementos de mesmo nome e aparecem em mais de um documento (Albinader e Lins, 2006).

Para resolver estes problemas, existe um recurso da *XML* chamado "*namespace*".

Um "*namespace*" é um grupo de nomes de elemento e atributo. Inserindo um prefixo de "*namespace*" a um nome de elemento ou atributo, o analisador *XML* saberá de que "*namespace*" ele veio e o validará com a respectiva DTD.

O nome do "*namespace*" é denominado prefixo de "*namespace*".

A sintaxe para qualificar o "*namespace*" de um elemento ou atributo está ilustrada na Figura G.7, onde a palavra "prefixo-ns" significa "prefixo de *namespace*". O prefixo de "*namespace*" (1) é associado por um sinal de dois-pontos (2) ao elemento ou atributo local (3).

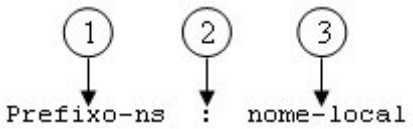


Figura G.7: Sintaxe para qualificar um "*namespace*" de um elemento ou atributo *XML*.

Para que se possa especificar o "*namespace*" de um elemento ou atributo, é preciso antes declará-lo. Esta declaração é feita através de um atributo dentro do elemento. A Figura G.8 mostra a sintaxe para declaração de um "*namespace*". A declaração é composta pelo prefixo de "*namespace*" (1) seguido pelo sinal de dois-pontos, pelo nome do elemento a ser qualificado (2), um espaço em branco, depois pela palavra-chave "xmlns" (3) seguida pelo sinal de dois-pontos, pelo prefixo de "*namespace*" (1), o sinal de igual e, finalmente, por um *URL*[3] entre aspas (4).

[3]O *URL* (Uniform Resource Locator) é formado pelo *URI* antecedido pelo meio de acesso (http: //, ftp: //, entre outros).

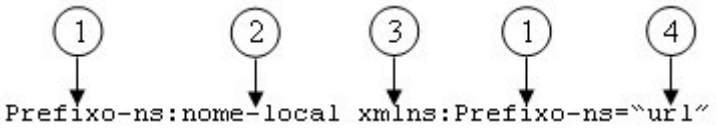


Figura G.8: Sintaxe para declarar um "*namespace*" *XML*.

A linha a seguir apresenta a declaração de um *namespace* para o elemento "formula".

```
<eq:formula xmlns:eq="http://www.mathstuff.org/">
```

As linhas a seguir apresentam um exemplo de qualificação de vários elementos de um documento *XML*:

```
<myns:journal xmlns:myns="http://www.psycholabs.org/mynamespace/">
    <myns:experiment>
        <myns:date> March 4, 2001</myns:date>
        <myns:subject> Effects of CAffeine on Psychokinetic Ability
        </myns:subject>
    </myns:experiment>
</myns:journal>
```

G.3.5 MODELAGEM DE DOCUMENTOS

A *XML* permite a criação livre de elementos e atributos. É possível definir formalmente uma linguagem em *XML* através de um processo chamado "modelagem de documento". Através desta modelagem, pode-se restringir o vocabulário dos elementos e atributos e controlar a gramática dos elementos.

O modelo é um tipo especial de documento, escrito em uma sintaxe criada para descrever linguagens *XML*, que estabelece explicitamente o vocabulário para uma única linguagem de marcação.

O tipo mais popular de modelo de documento é a DTD (*Document Type Definition*).

Uma DTD é um arquivo texto composto de uma sequência de declarações que definem um tipo de documento da seguinte maneira:

– Declaração de um conjunto de elementos permitidos (vocabulário). Não se pode utilizar nenhum nome de elemento além do permitido neste conjunto;

– Definição de um modelo de conteúdo para cada elemento (gramática). O modelo de conteúdo é um padrão que diz quais elementos ou dados podem entrar em um elemento, em que ordem, em que quantidade e se eles são obrigatórios ou opcionais;

– Declaração de um conjunto de atributos permitidos para cada elemento. Cada declaração de atributo define o nome, tipo de dados, valores pré-definidos e comportamento (obrigatório ou não) do atributo;

– Ela oferece uma série de mecanismos para facilitar o gerenciador do modelo, por exemplo, o uso de entidades de parâmetro e a capacidade de importar partes do modelo a partir de um arquivo externo.

Maiores detalhes sobre a especificação de um modelo de documento DTD podem ser encontrados em Ray (2001).

Outro tipo de modelo de documento é o *XML Schema*. Ao contrário da DTD, ele é escrito em *XML* e oferece muito mais controle sobre os tipos e padrões de dados, tornando-o uma linguagem mais atraente para impor requisitos escritos de entrada de dados.

Um documento *XML* é associado a um *XML Schema* no seu prólogo na declaração de tipo de documento. O trecho de código *XML* a seguir exemplifica esta associação para o elemento raiz "*note*":

```
<?xml version="1.0"?>
<note  xmlns="http://www.w3schools.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3schools.com/note.xsd">
.....
</note>
```

Um documento *Xml Schema* é identificado como tal na primeira linha, que o associa ao *namespace* da *XML Schema*:

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
```

Em seguida, vem um bloco especial "*annotation*", que é um local para documentar a finalidade do esquema e outros detalhes:

```
<xsd:annotation>
    <xsd:documentation>
        Census form for the Republic of Oz
    </xsd:documentation>
</xsd:annotation>
```

Após estes dois blocos iniciais, vem as declarações de elementos e atributos.

A sintaxe de uma declaração básica de elemento é mostrada na Figura G.9. Ela é composta pela seqüencia de caracteres (marca) "$< xsd : element$" (1) seguido pelos atributos name (2) cujo valor é o nome do elemento a ser modelado e pelo atributo type (3) cujo valor indica o tipo do elemento. Finalmente, a seqüencia de caracteres "/>" encerra a declaração.

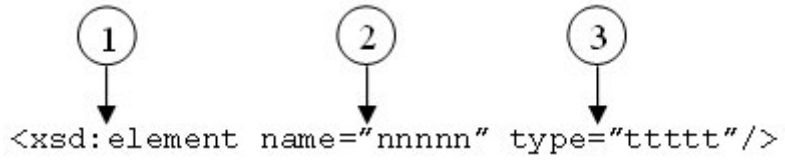


Figura G.9: Sintaxe de uma declaração básica de elemento para a *XML Schema*.

Os elementos podem ser de tipo simples ou complexos. Os elementos são ditos de tipo simples quando não possuem atributos ou outros elementos. Por exemplo, para declarar o elemento ***firstname*** do tipo ***string*** usa-se a forma:

```
<xsd:element name="firstname" type="xsd:string"/>
```

Alguns tipos predefinidos pela *XML Schema* são:

(i) byte, float, long, decimal, INF (infinito): formatos numéricos.

(ii) time, date, timeinstant, timeduration: padrões para marcar hora, data e duração.

(iii) boolean: representa um valor true (verdadeiro) ou false (false).

(iv) ID, IDREF, IDREFS, NMTOKEN, NMTOKENS: tipos de atributo que funcionam exatamente como seus correspondentes nas DTDs.

Os elementos complexos recebem um nome para o tipo que será descrito posteriormente.

Exemplo de tipo complexo:

```
<xsd:element name="address" type="Address"/>
...
<xsd:complexType name="Address">
    <xsd:element name="number" type="xsd:decimal"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
</xsd:complexType>
```

É possível criar novos tipos para restringir os valores dos dados. Por exemplo, para restringir os valores de um elemento ***age*** a números positivos inferiores a 200, pode-se declarar o elemento ***age*** da seguinte maneira:

```
<xsd:element name="age">
    <xsd:simpleType base="xsd:positive-integer">
        <xsd:maxExclusive value="200"/>
    </xsd:simpleType>
</xsd:element>
```

A especificação completa do *XML Schema* pode ser encontrada na recomendação da W3C (Fallside e Walmsley, 2004).

G.4 WSDL

A *WSDL* (*Web Service Description Language*) é uma linguagem que estabelece um modelo e um formato *XML* para descrever um Serviço Web.

Um documento *WSDL* é um arquivo em formato *XML* padronizado que descreve o serviço remoto de maneira estruturada. Ele descreve a interface do serviço, os tipos de dados usados e onde o serviço está localizado.

A estrutura de um documento *WSDL* possui seis principais elementos, os quais estão representados na Figura G.10 e discutidos a seguir:

– Elemento **<definitions>**: é o elemento raiz do documento *WSDL*. Ele define o nome do Serviço Web, declara os *namespaces* utilizados e contém todos os outros elementos do documento;

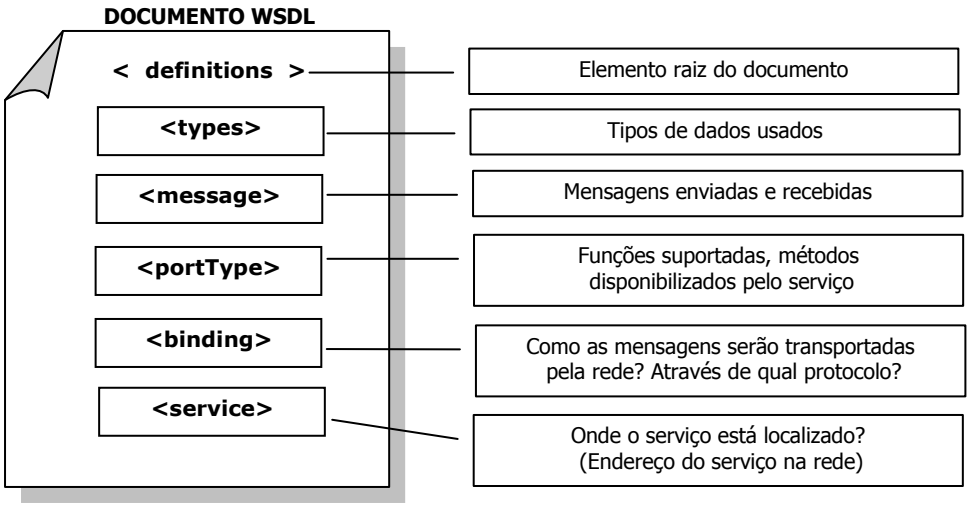


Figura G.10: Estrutura de uma mensagem *WSDL*.

– Elemento **<types>**: especifica e lista todos os tipos usados pelo Serviço Web. A especificação *WSDL* herda o conjunto de tipos padrão do *XML Schema*. Assim, se o serviço não usa nenhum tipo além dos definidos no *XML Schema*, o elemento <types> não é necessário;

– Elemento **<message>**: todas as mensagens que possam vir a ser trocadas entre o consumidor do serviço e o serviço são listadas no documento *WSDL* através de elementos <message>. Cada elemento <message> possui um atributo *name*, que define o nome da mensagem, e pode conter elementos <part>, que especificam os atributos da mensagem;

– Elemento **<portType>**: é a interface abstrata do Serviço Web. É um conjunto de operações e mensagens abstratas envolvidas (Christensen et al., 2001). Cada elemento <portType> possui um atributo *name*, que define o seu nome, e contém elementos <operation>. O elemento <operation> define o tipo de interação entre o consumidor e o serviço e o associa a um elemento <message> definido. Esta interação pode ser:

  * mensagem em um único sentido: o serviço recebe a mensagem e nenhuma resposta é retornada. O elemento <operation> conterá o elemento <input>;

  * solicitação-resposta: o consumidor envia uma mensagem ao servidor. O servidor

responde com uma mensagem contendo o resultado da solicitação ou uma mensagem de falha. O elemento <operation> conterá o elemento <input>, o elemento <output> e, opcionalmente, o elemento <fault>;

* resposta-solicitação: a comunicação ocorre no sentido inverso ao do tipo solicitação-resposta. O serviço envia uma mensagem ao cliente e recebe deste uma resposta. O elemento <operation> conterá primeiro o elemento <output>, depois o elemento <input> e, opcionalmente, o elemento <fault>. Para que este tipo de interação ocorra, o Serviço Web precisa conhecer o cliente que vai responder à solicitação e isto pode acontecer através de uma inscrição prévia deste cliente ao serviço;
* notificação: o Serviço Web envia uma mensagem ao cliente para notificá-lo sobre a ocorrência de alguma espécie de evento no qual o cliente registrou-se previamente. Neste tipo de interação, o elemento <operation> conterá apenas o elemento <output>.

– Elemento **<binding>**: associa protocolos específicos para a invocação do serviço aos elementos <operation> e <message> definidos no elemento <portType>. O elemento <binding> possui um atributo *name* que define seu nome e o atributo *type* que liga o elemento <binding> a respectiva *portType* e suas operações. Atualmente, a especificação *WSDL* define extensões de ligação do elemento <binding> com pontos de acesso capazes de entender os protocolos *SOAP* transportados sobre: *HTTP*, *HTTP GET/POST* e *MIME* (Albinader e Lins, 2006). Um elemento <binding> deve especificar apenas um protocolo (Christensen et al., 2001);

– Elemento **<service>**: define o endereço de rede do serviço a ser invocado. Ele pode conter um ou mais elementos <port>, que são quem realmente contém o endereço do Serviço Web. Para cada elemento <binding> deve haver um elemento <port>.

Além deste principais elementos, a especificação *WSDL* define ainda outros elementos úteis, tais como:

– Elemento **<documentation>**: elemento usado para fornecer documentação e pode ser usado dentro de qualquer elemento *WSDL*;

– Elemento **<import>**: possibilita o referenciamento entre dois documentos *WSDL* e o estabelecimento do *namespace* para os elementos importados. A reutilização constitui uma das prioridades da tecnologia *WSDL* (Albinader e Lins, 2006).

G.5 Axis

O *Axis*2, um projeto da Apache (`http://www.apache.org`), é uma implementação do protocolo *SOAP* (ver seção 4.4) utilizado em Serviços Web (Axis2, 2007).

O *Axis* oferece uma série de utilitários que facilitam o desenvolvimento dos Serviços Web, como geradores de código a partir de descritores *WSDL* e vice-versa e geradores de arquivos ".aar" para *deploy*.

As ferramentas *WSDL2Java*, que gera código Java a partir do arquivo *WSDL* e *Java2WSDL* que gera o documento *WSDL* a partir de uma interface (ou classes) Java são especialmente úteis no desenvolvimento de Serviços Web.

A Tabela G.1 mostra, simplificadamente, o funcionamento da ferramenta *WSDL2Java* (Axis2, 2007).

É importante observar que o *Skeleton* gerado apresenta o corpo do código vazio, pois cabe ao desenvolvedor acrescentar toda a lógica do negócio, ou seja, o código necessário para a implementação do Serviço Web. A classe *Stub* para ser utilizada por clientes, entretanto, é gerada completa, não necessitando de nenhuma codificação adicional. Basta que os arquivos gerados sejam copiados para o diretório do restante das classes da aplicação cliente. Esta classe é a responsável por fazer com que a invocação do Serviço Web se comporte como uma chamada de método local.

Para a utilização das classes geradas, a aplicação cliente necessitará indicar no seu "*classpath*" (variável que informa ao compilador o caminho para arquivos adicionais necessários para

Tabela G.1: Criação de classes Java a partir do *WSDL* através do *Axis*2 (Axis2, 2007).

Caso de Uso	**Classe(s) Java gerada(s)**
Para cada entrada na seção **<types>** do *WSDL*. PARA O **CLIENTE** OU **SERVIDOR**	- Uma classe Java;
Para o lado do **SERVIDOR**.	- Uma classe Java, chamada *Skeleton*: ****Skeleton.java**; - Uma classe Java auxiliar: ***MessageReceiverInOut.java**.
Para o lado do **CLIENTE**.	- Uma classe Java, chamada *Stub*: ****Stub.java**; - Uma classe Java auxiliar: ****CallbackHandler.java**.

a compilação ou execução da aplicação) a biblioteca *Axis*, que é formada por alguns arquivos de extensão ".jar" (para linguagem Java) e pode ser obtida no *site* do projeto (Axis2, 2007).

Apêndice H

Detalhamento das Tecnologias Relacionadas a Aplicações Web

H.1 Introdução

Neste apêndice são apresentados maiores detalhes sobre as tecnologias relacionadas a Aplicações Web.

H.2 HTML

HTML é a sigla para "*HyperText Markup Language*" ou, em português, linguagem de marcação para hipertexto. Ela é a linguagem usada na construção de páginas Web.

A Tabela H.1 mostra uma lista das marcas *HTML* mais usadas. Para maiores detalhes, o "*site*" da W3C (*The World Wide Web Consortium*) - `http://www.w3.org/MarkUp/` contém a especificação completa e uma série de referências para tutoriais e outras informações sobre o assunto.

Tabela H.1: Marcas mais usadas em documentos *HTML*.

Marca	Descrição
<HTML> ... </HTML>	Delimitam um documento *HTML*. Todo conteúdo da página deve estar contido dentro destas marcas.

continua ...

Tabela H.1: Marcas mais usadas em documentos *HTML*.

Marca	**Descrição**
<HEAD> ... </HEAD>	Delimitam o cabeçalho do documento. Contém informações para o navegador, como título, palavras-chave, etc.
<title> ... </title>	Título que aparecerá na barra superior do navegador. Deve estar posicionado dentro do bloco <HEAD>..</HEAD>
<BODY> ... </BODY>	Delimita a região que realmente será exibida pelo navegador.
 ... 	Faz com que o texto ou imagem contidos entre as marcas seja visualizado como "*link*", ou seja, quando "clicado", o navegador é redirecionado para outra página, cujo endereço deve estar especificado em "end...".
...	Cria uma lista sem numeração. Cada item da lista deve estar após a marca .
<img src="nome.ext" >	Insere a figura que está no arquivo "nome.ext" na página.
<table> ... </table>	Cria uma tabela.
<tr> ... </tr>	Define uma linha da tabela.
<td> ... </td>	Define uma célula da linha da tabela.
...	Formata o texto em negrito.
...	Centraliza o texto.
 	Insere uma quebra de linha.
...	Formata o texto em itálico.

continua ...

Tabela H.1: Marcas mais usadas em documentos *HTML*.

Marca	Descrição
...	Estilo de título: Fonte de tamanho grande e em negrito. Quanto maior o número após a letra h menor o tamanho da fonte.
 ... 	Altera as características (atrib) do texto (fonte, tamanho, cor, etc.).

H.3 Tapestry

O *Tapestry* é uma poderosa biblioteca Java utilizada em aplicações Web para geração dinâmica de conteúdo.

H.3.1 Componentes Tapestry

Um aplicativo Web é formado por um conjunto de páginas *HTML* construídas com o uso de componentes. O componente é responsável por gerar dinamicamente conteúdo de partes da página Web, com o uso de classes Java. O *Tapestry* oferece por volta de 50 componentes, que são suficientes para atender às necessidades de um desenvolvedor Web (Smart, 2006). É possível, ainda, desenvolver novos componentes *Tapestry* para serem utilizados nas aplicações Web.

Os componentes *Tapestry* são identificado no código *HTML* da página modelo, pela palavra ***jwcid*** e podem ser do tipo implícitos ou declarados:

– **Componentes implícitos:** são aqueles que descrevem o seu tipo e seus parâmetros diretamente no código *HTML*. A sintaxe básica deste tipo de componente está ilustrada na Figura H.1. O componente é inserido em uma marca *HTML* (1) e é composto pela palavra **jwcid** (2), seguido do sinal de igual, do sinal de aspas, do nome de seu identificador (3),

do símbolo @ (4), do seu tipo (5), do sinal de aspas e de um ou mais parâmetros. Estes parâmetros vêm separados por espaços em branco e são formados pelo nome do parâmetro (6), seguido pelo sinal de igual (7), pelo sinal de aspas, pelo valor associado ao parâmetro (8) e outro símbolo de aspas. O sinal de maior (9) fecha a marca *HTML*. O nome do identificador não é obrigatório e, quando ele não é fornecido, o *Tapestry* cria um nome para ele automaticamente.

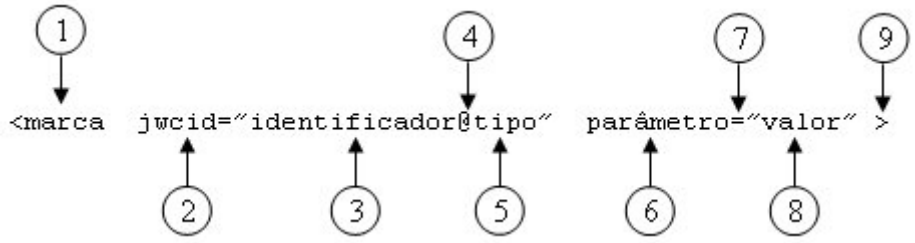


Figura H.1: Sintaxe de um componente implícito do *Tapestry*.

No exemplo seguinte, é definido um componente (sem identificação) do tipo de inserção (*Insert*) cujo parâmetro "*value*" terá como valor o conteúdo recuperado do atributo "*price*" da classe Java "`feature.Destination`". Este valor substituirá o número 199 pois o mesmo está envolvido pela marca <*span*>. O prefixo *ognl* indica que o que vem após os dois pontos é uma expressão *OGNL* (*Object Graph Navigation Language*), a qual informa ao *Tapestry* que o valor "`featureDestination.price`" é uma expressão a ser avaliada e não uma "*String*" literal.

```
<span jwcid="@Insert" value="ognl:featureDestination.price">199</span>
```

– **Componentes declarados:** são aqueles que têm o seu tipo e parâmetros definidos externamente, em um arquivo de especificações. A sintaxe deste tipo de componente está mostrada na Figura H.2. O componente também é inserido em uma marca *HTML* (1) e é formado apenas pela palavra **jwcid**(2), seguido do sinal de igual, do sinal de aspas, do nome de seu identificador (3), do sinal de aspas e pelo sinal de maior (4) que fecha a marca *HTML*.

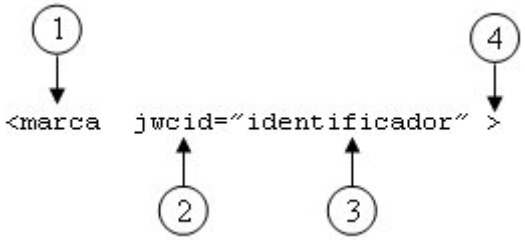


Figura H.2: Sintaxe de um componente declarado do *Tapestry*.

O arquivo de especificações de componentes *Tapestry* tem o mesmo nome do modelo *HTML* seguido pela extensão ".page". Ele é um arquivo *XML* com a seguinte forma:

```
<?xml version="1.0"?>
<!DOCTYPE page-specification PUBLIC
    "-//Apache Software Foundation//Tapestry Specification 4.0//EN"
    "http://jakarta.apache.org/tapestry/dtd/Tapestry_4_0.dtd">
<page-specification class="contexto.Home">

</page-specification>
```

As linhas iniciais do arquivo *XML* formam o prólogo e identificam o documento como sendo um modelo de documento público *Tapestry*. O bloco seguinte, formado pelo elemento raiz <page-specification> é onde devem ser inseridas as especificações dos componentes declarados. O atributo "*class*" identifica a classe Java principal. Seu valor deve ser o contexto (caminho de diretórios onde estão armazenadas as classes Java da aplicação) seguido por um ponto e pelo nome da classe, que tem o mesmo nome do arquivo *HTML* e do arquivo de especificações (neste exemplo a classe usada foi `Home.class`).

A lista a seguir mostra alguns exemplos de componentes *Tapestry* (Tapestry, 2006):

– **Insert:** usado para inserção de texto no código `HTML` final.

– **Image:** usado para exibir uma imagem. Por exemplo, o componente *Image* definido no código *HTML* a seguir:

```
<img jwcid="@Image" image="asset:imageAsset" alt="View Tapestry Home"/>
```

Sua declaração no arquivo "`.page`" pode ser feita através do elemento "<*asset*>". Este elemento é um tipo especial de componente para referenciar um arquivo externo.

```
<asset name="imageAsset" path="/images/poweredby.png"/>
```

– **Form:** usado para obter dados do usuário. É associado à marca *HTML* "<*form*>" que é um bloco composto de outros elementos usados em formulários *HTML*. O exemplo abaixo mostra um formulário *HTML* associado ao componente do tipo "*Form*" do *Tapestry*:

```
<form jwcid="stockQuoteForm">
    <input type="text" jwcid="stockId"/>
    <input type="submit" value="OK"/>
</form>
```

Este componente pode ser declarado no arquivo de especificações da seguinte maneira:

```
<component id="stockQuoteForm" type="Form">
    <binding name="listener" value="listener:onOk"/>
</component>
```

O valor do parâmetro "*listener*" referencia o nome do método que é acionado quando o usuário "clica" no botão de envio de dados na página (normalmente é um botão com a indicação *OK* ou *ENVIAR*).

– **TextField:** associado ao elemento de formulário *HTML* chamado "<*input*>" , que é um campo para digitação de dados. No exemplo *HTML* mostrado no item anterior foi definido o componente de nome "*stockId*" que pode ser declarado no arquivo de especificações da seguinte maneira:

```
<component id="stockId" type="TextField">
    <binding name="value" value="ognl:stockId"/>
</component>
```

– **PropertySelection:** associado ao elemento de formulário *HTML* "<*select*>", que apresenta uma lista de opções ao usuário. Em um código *HTML* ele pode ser definido do seguinte modo:

```
<select jwcid="stockId">
    <option value="0">IBM</option>
    <option value="1">RHAT</option>
</select>
```

Os elementos "<*option*>" definem as opções a serem apresentadas. O componente "*stockId*" pode ser declarado no arquivo de especificações da seguinte maneira:

```
<component id="stockId" type="PropertySelection">
    <binding name="model" value="availStockIds"/>
    <binding name="value" value="stockId"/>
</component>
```

O parâmetro "*model*" armazena as opções a serem exibidas pelas marcas "<*option*>" no *HTML*. Estas opções serão obtidas pelo método *getAvailStockIds()* da classe Java em questão. O parâmetro "*value*" indica onde será armazenada a opção escolhida pelo usuário, que no exemplo é o atributo "*stockId*" da classe Home.java.

– **DatePicker:** permite que o usuário forneça uma data "clicando" diretamente em um calendário gráfico na página. O componente "quoteDate" definido no trecho de código *HTML* abaixo, ilustra este tipo de componente:

```
<span jwcid="quoteDate">May 3, 2005</span>
```

Este componente pode ser declarado no arquivo de especificações da seguinte maneira:

```
<component id="quoteDate" type="DatePicker">
    <binding name="value" value="quoteDate"/>
</component>
```

Outros tipos de componentes, assim como seus tipos de parâmetros válidos, podem ser encontrados no "*site*" oficial do *Tapestry* (Tapestry, 2006).

H.3.2 Outros recursos Tapestry

O *Tapestry* oferece muitos recursos que facilitam e simplificam a programação da aplicação Web. A seguir são mostrados alguns destes recursos:

– **Property:** Este recurso, permite que o *Tapestry* crie e gerencie um atributo de uma classe Java que guarde um valor que será usado no programa. O *Tapestry* extende a classe onde o "*Property*" é declarado (no arquivo .page) e cria os métodos *getAtributo()*, *setAtributo(tipo variável)* e *initialize()*. Se alguma classe Java precisar acessar algum destes métodos, basta que a assinatura do método seja declarada como "*abstract*". O *Tapestry* será o responsável por implementar este método. O componente "*stockId*" mostrado no exemplo de componente "*TextField*" pode ser definido como "*Property*" através da seguinte declaração no arquivo de especificações (de extensão .page):

```
<property name="stockId"/>
```

Na classe Java correspondente, o método para recuperar o valor deste atributo, pode ser definido do seguinte modo:

```
abstract public String getStockId();
```

O tipo de retorno do método informa ao *Tapestry* de que tipo será este atributo, no caso uma "*String*".

– **Validação de dados:**

O *Tapestry* oferece alguns recursos simples para validação de dados digitados pelo usuário. Por exemplo, um componente do tipo "*TextField*" pode ser definido para aceitar apenas números. Para isso, basta acrescentar um parâmetro (*binding*) de nome "*translator*" ao componente no arquivo de especificações. Por exemplo, para um componente de nome "*weight*" do tipo "*TextField*" pode-se restringir seus valores a tipos numéricos, da seguinte maneira:

```
<component id="weight" type="TextField">
    <binding name="value" value="weight"/>
    <binding name="translator" value="translator:number"/>
</component>
```

Se um valor inválido é digitado, é retornada uma mensagem de erro. O *Tapestry* oferece alguns tipos predefinidos para validação e permite que se crie novos tipos.

Além do parâmetro "*translator*", o parâmetro "*validators*" pode ser usado para definir o comportamento do dado. No exemplo abaixo:

```
<component id="weight" type="TextField">
    <binding name="value" value="weight"/>
    <binding name="translator" value="translator:number"/>
    <binding name="validators" value="validators:required,min=0"/>
</component>
```

O componente "*weight*" é definido como obrigatório pelo atributo "*value*" através da palavra "*required*". Além disso, foi definido que o valor deve ser maior que zero no atributo "*value*" através da expressão "min=0".

O *Tapestry* permite agilizar esta validação, permitindo o uso de *JavaScripts*[1]. Assim, a validação é feita no lado do cliente. Para isso, pode-se usar o parâmetro "*clientValidationEnabled*" no componente do tipo "*Form*", como mostrado no exemplo a seguir:

```
<component id="form" type="Form">
    <binding name="listener" value="listener:onSubmit"/>
    <binding name="clientValidationEnabled" value="true"/>
</component>
```

H.4 XSL

A *XSL* (*Extensible Stylesheet Language*) ou Linguagem de Folhas de Estilo Extensível é uma família de recomendações para transformação e apresentação de documentos *XML*.

A XLS se baseia em três componentes, segundo as recomendações da W3C (`http://www.w3.org/Style/XSL/`), cujas principais características serão apresentadas a seguir.

[1] *JavaScript* é uma linguagem de programação simples muito usada em páginas *HTML* que é interpretada pelos navegadores.

H.4.1 XPath

A *XPath* ou "*XML Path Language*" é uma linguagem para acessar e referenciar elementos e atributos em um documento *XML*. Estas referências acontecem através de expressões de caminho, parecidas com as expressões de sistemas de arquivos.

A expressão *XPath* abaixo seleciona o elemento raiz (*catalog*) de um documento *XML*.

```
/catalog
```

Já expressão *XPath* a seguir seleciona todos os elementos *price* de todos os elementos *cd* do elemento *catalog*.

```
/catalog/cd/price
```

A *XPath* também define uma biblioteca de funções para manipulação de textos (cadeias de caracteres) e números, além de expressões booleanas. O exemplo a seguir, seleciona todos os elementos *cd* que têm um elemento *price* cujo valor seja maior que 10.80.

```
/catalog/cd[price>10.80]
```

É permitido o uso de curingas (representados pelo caractere *) nas expressões de caminho. A expressão a seguir, seleciona todos os elementos filhos de todos os elementos *catalog*.

```
/catalog/*
```

O uso de duas barras indica que a seleção pode ocorrer em qualquer nível da árvore *XML*, como mostrado na expressão a seguinte, que seleciona todos os elementos *cd* do documento *XML*.

```
//cd
```

Os atributos de elementos *XML* são referenciados no *XPath* pelo caractere "@". A expressão seguinte, seleciona todos os elementos *cd* que têm um atributo chamado *country* com valor 'UK'.

```
//cd[@country='UK']
```

As expressões podem ser relativas ao nó de análise atual, como a expressão a seguir, que seleciona todos os elementos filhos do elemento atual.

```
child::*
```

A Tabela H.2 mostra alguns exemplos de funções *XPath*.

Tabela H.2: Exemplos de funções *XPath*.

Função	Descrição	Uso
count()	Retorna o número de nós de um conjunto de nós.	count(child::cd)
last()	Retorna o número da posição do último nó na lista de nós processados.	position()=last()
concat()	Retorna a concatenação de textos.	concat('The',' ','XML') Resultado: 'The XML'
starts-with()	Retorna verdadeiro se a primeira string começa com a segunda string, senão retorna falso.	starts-with('XML','X') Resultado: true
round()	Arredonda o argumento para o inteiro mais próximo.	round(3.14) Resultado: 3

H.4.2 XSLT

A *XSLT* ou "*XSL Transformations*" é uma linguagem para transformações de dados *XML*. Com esta linguagem, é possível transformar documentos *XML* em outros tipos de documentos, como documentos que são reconhecidos pelos navegadores (*HTML* ou *XML*) ou documentos em formato texto comum, PDF, entre outros. Durante a transformação, a *XSLT* permite filtrar dados, reordená-los, testar, adicionar novas informações, além de outras possibilidades.

As regras *XSLT* de transformação são definidas em um arquivo *XSL*, que é um arquivo texto escrito em *XML*, no qual expressões *XPath* são usadas para montar os padrões de transformação. Neste arquivo, as regras são montadas em partes chamadas de moldes. Durante a transformação, quando os dados *XML* encontram uma combinação com os moldes, o resultado é produzido.

Um arquivo *XML* é identificado como *XSL* quando o elemento raiz é a palavra "*stylesheet*" ou "*transform*", além de ter declarado seu *namespace* de acordo com as recomendações W3C, como mostrado no trecho de código abaixo:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

No restante do documento *XSL* são declarados os moldes, que contém as regras a serem aplicadas ao documento *XML*. Um molde é definido usando-se o elemento <xsl:template>. No trecho de código *XSL* a seguir, o elemento template será aplicado a todos os elementos *XML*, pois a expressão *XPath* usada foi:

```
<xsl:template match="/">
```

O símbolo da barra sozinho indica que o molde será válido em todos os elementos *XML*. Observa-se os elementos *HTML* adicionados (<html>,<body>,<h2>, entre outros) e as primeiras regras de filtragem definidas:

```
<td><xsl:value-of select="catalog/cd/title"/></td>
<td><xsl:value-of select="catalog/cd/artist"/></td>
```

Nesta duas linhas, o elemento <xsl:value-of> extrai os valores dos elementos *tittle* e *artist*. A extração ocorre através das expressões *XPath* mostradas acima. Para que todos os elementos *cd* sejam avaliados, o elemento de laço <xsl:for-each> é utilizado e uma expressão *XPath* limita esta avaliação aos elementos *cd*:

```
<xsl:for-each select="catalog/cd">
```

Alguns elementos *XSLT* são mostrados na Tabela H.3. A lista completa dos elementos *XSLT* pode ser encontrada em w3Schools (2007*b*).

Tabela H.3: Exemplos de elementos *XSLT*.

Elemento	**Descrição**	**Exemplo**
<xsl:for-each>	Permite avaliar todos os elementos relacionados pela expressão *XPath* select="...".	<xsl:for-each select="catalog/cd">

continua ...

Tabela H.3: Exemplos de elementos *XSLT*.

Elemento	**Descrição**	**Exemplo**
<xsl:sort>	Permite definir um elemento para ordenação dos resultados.	<xsl:sort select="artist"/>
<xsl:if>	Executa as ações definidas somente se uma condição for verdadeira.	<xsl:if test="price > 10"> (se o valor do elemento price for maior que 10)
<xsl:choose>	Usado em conjunto com os elementos <xsl:when> e <xsl:otherwise> para expressar múltiplos testes condicionais.	<xsl:choose> <xsl:when test="price > 10"> ... </xsl:when> <xsl:otherwise> ... </xsl:otherwise> </xsl:choose>
<xsl:apply-templates>	Permite aplicar uma regra de molde ao elemento corrente ao aos seus filhos. O molde é definido em outro bloco no arquivo.	... <xsl:apply-templates/> .... <xsl:template match="cd"> <xsl:apply-templates select="title"/> <xsl:apply-templates select="artist"/> </xsl:template>

H.4.3 XSL-FO

A *XSL-FO* ou "*XSL Formatting Objects*" é uma linguagem para formatação de dados *XML* para exibição em telas, papéis e outras mídias. Com ela é possível definir o *layout* de página, fontes, estilos, cores, imagens e muitas outras propriedades.

Um documento *XSL-FO* é composto por:

(i) Um cabeçalho *XML* e uma declaração de namespace apropriada: `<fo:rootxmlns:fo="http://www.w3.org/1999/XSL/Format">`;

(ii) Informações de *layout* de página;

(iii) Definições de cabeçalhos e rodapés;

(iv) Conteúdo (texto).

A Figura H.3 mostra o esquema do *layout* de página do *XSL-FO* que é composta por cinco regiões: o corpo (*region-body*), anterior (*region-before*), posterior (*region-after*), início (*region-start*) e fim (*region-end*).

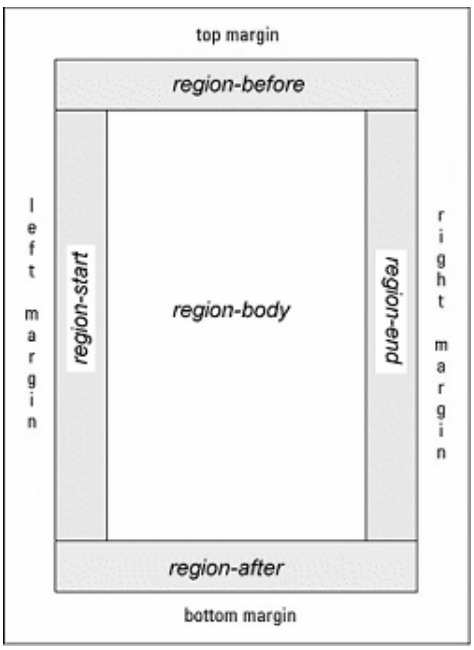


Figura H.3: Esquema do *layout* de página do *XSL-FO* (w3Schools, 2007a).

A Tabela H.4 mostra alguns elementos *XSL-FO*. A lista completa destes elementos pode ser encontrada em w3Schools (2007*a*).

Tabela H.4: Exemplos de elementos *XSL-FO*.

Elemento	**Descrição**	**Exemplo**
<fo:root>	É o elemento principal de um documento *XSL-FO*.	<fo:root> ... outros elementos... </fo:root>
<fo:layout-master-set>	Elemento contêiner de elementos <fo:simple-page-master> que definem os *layouts* de página.	<fo:layout-master-set> ... </fo:layout-master-set>
<fo:simple-page-master>	Elemento de definição de *layouts* de página (tamanho, margens, entre outros).	<fo:simple-page-master master-name="A4"> ... </fo:simple-page-master>
<fo:page-sequence>	Elemento que descreve regiões de conteúdo efetivo de uma página associados a um *layout* pelo seu nome.	<fo:page-sequence master-reference="A4"> ... </fo:page-sequence>
<fo:flow>	Elemento que descreve uma região de conteúdo de uma página.	<fo:flow flow-name="PageBody" font-size="12pt"> ... </fo:flow>

continua ...

Tabela H.4: Exemplos de elementos *XSL-FO*.

Elemento	**Descrição**	**Exemplo**
<fo:block>	Elemento que define partes do texto, como parágrafos, tabelas, listas, entre outros.	<fo:block border-width="1mm"> ... texto ... </fo:block>
<fo:footnote>	Elemento que define uma nota de pé de página.	<fo:footnote> ... texto ... </fo:footnote>
<fo:list-block>	Elemento que define uma lista, cujos ítens são definidos por elementos <fo:list-item>.	<fo:list-block> <fo:list-item> ... texto </fo:list-item> </fo:list-block>
<fo:page-number>	Elemento usado para representar o número de página corrente.	<fo:page-number> ... </fo:page-number>
<fo:float>	Elemento usado para posicionar uma imagem em uma determinada área.	<fo:float> ... </fo:float>

continua ...

Tabela H.4: Exemplos de elementos *XSL-FO*.

Elemento	**Descrição**	**Exemplo**
<fo:table>	Elemento usado para formatar dados em forma de tabela. Contém um elemento <fo:table-body>, um <fo:table-row> para cada linha, um <fo:table-cell> para cada célula entre outros elementos.	<fo:table> <fo:table-body> <fo:table-row> <fo:table-cell> ... texto... </fo:table-cell> </fo:table-row> </fo:table-body> </fo:table>

Apêndice I

Glossário

A

API: sigla para "*Application Programming Interface*", uma interface escrita em alguma linguagem de programação, que pode ser compilada e que permite a utilização de outros programas ou funções;

AXIOM: sigla para "*AXIs Object Model*", uma API desenvolvida pela Apache para processamento de arquivos XML;

AXIS: uma implementação do protocolo SOAP (em Java ou C++), usada em Serviços Web.

C

CORBA: sigla para "*Common Object Request Broker Architecture*" um padrão aberto que possibilita computação distribuída em ambientes de aplicações heterogêneos;

CSS: sigla para "*Cascading Style Sheets*" ou folha de estilo em cascata, uma linguagem para formatação de páginas Web.

D

DCOM: sigla para "*Distribuited Common Object Model*", uma tecnologia da Microsoft para a computação distribuída na plataforma *Windows*, que utiliza RPC;

Deploy: termo que significa instalação ou distribuição de um sistema de software, mais usado em aplicações Web;

Download: ato de descarregar ou copiar arquivos através de um navegador Web, para o computador do usuário.

E

E-Mail: correio eletrônico ou mensagem enviada através da Internet.

J

JavaScript: linguagem de programação simples muito usada em páginas Web;

J2EE: sigla para "*Java 2 Enterprise Edition*", um modelo de programação baseado na Web e em componentes de negócios, baseado em Java.

H

HTML: sigla para "*HyperText Markup Language*" ou linguagem de marcação para hipertexto com a qual as páginas Web são feitas;

HTTP: sigla para "*Hypertext Transfer Protocol*", protocolo para envio e recebimento de dados pela Internet.

L

Link: texto exibido em uma página Web que, ao ser "clickado", redireciona o navegador para uma outra página.

M

MOM: sigla para "*Message-Oriented Middleware*", um modelo para comunicação indireta entre aplicações, baseado em mensagens.

P

Plug-in: programa ou recurso que adiciona uma funcionalidade a outro programa.

R

RIA: sigla para "*Rich Internet Applications*", uma aplicação Web com características e funcionalidades de uma aplicação desktop tradicional;

RMI: sigla para "*Remote Method Invocation*", uma tecnologia Java para comunicação remota entre aplicações Java;

RPC: sigla para "*Remote Procedure Calls*", tecnologia para comunicação entre aplicações remotas.

S

Servlets: classes Java escritas para serem executadas em um servidor;

Skeleton: implementação localizada no lado do servidor usado em Serviços Web;

Site: coleção de páginas Web que podem ser acessadas pela Internet;

SOAP: sigla para "*Simple Object Access Protocol*", protocolo para mensagens trocadas entre serviços Web e aplicações cliente, baseado em XML;

Stub: implementação localizada no lado do cliente que encapsula a comunicação com os Serviços Web;

Swing: biblioteca Java de geometria computacional que oferece poderosos recursos gráficos.

T

Tapestry: biblioteca Java utilizada para geração de conteúdo dinâmico em aplicações Web;

TCP/IP: sigla para "*Transmission Control Protocol/Internet Protocol*", um protocolo de rede usado na Internet;

Tomcat: servidor de aplicações Java para Web.

U

UDDI: sigla para "*Universal Description, Discovery and Integration*", um padrão para registros baseados em XML que contêm informações sobre Serviços Web;

Upload: ato de enviar arquivos através de um navegador Web, para um computador situado em algum ponto da Internet;

URL: sigla para "*Uniform Resource Locator*", que designa um endereço Web, formado pelo URI antecedido pelo meio de acesso (http: //, ftp: //, entre outros);

URI: sigla para "*Uniform Resource Identifier*", que designa uma cadeia de caracteres que identifica um recurso na Web.

X

XML: sigla para "***Ex****tensible* ***M****arkup* ***L****anguage*" ou, em português, Linguagem de Marcação Extensível, utilizada para armazenamento de dados;

XPath: sigla para "*XML Path Language*", um dos componentes da familia XSL, usado para acessar e referenciar elementos e atributos em um documento XML;

XSL: sigla para "***E****xtensible* ***S****tylesheet* ***L****anguage*" ou, em português, Linguagem de Folha de Estilos Extensível, utilizada para transformação de documentos XML;

XSL-FO: sigla para "*XSL Formatting Objects*", um dos componentes da familia XSL responsável pela formatação de dados XML para exibição em mídias específicas;

XSLT: sigla para "*XSL Transformations*", também componente da família XSL responsável por realizar transformações de dados XML.

W

WSDL: sigla para "*Web Service Description Language*", linguagem padronizada para descrição de Serviços Web.

Referências Bibliográficas

Albinader, J. A. N. e Lins, R. D., 2006. *Web services em Java*.

AXIOM, A. S. F., 2007, 'Apache axiom', URL: (http://ws.apache.org/commons/axiom/index.html). Última Consulta: 07/2007.

Axis2, A. S. F., 2007, 'Apache axis2 user's guide', URL: http://ws.apache.org/axis2/1_2/userguide.html. Última Consulta: 07/2007.

Bellwood, T., 2002. 'Uddi version 2.04 api specification'. *UDDI Committee Specification*, . URL: http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. e Orchard, D., 2004. 'Web services architecture'. *W3C Working Group Note*, . URL: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

Christensen, E., Curbera, F., Meredith, G. e Weerawarana, S., 2001. 'Web services description language (wsdl) 1.1'. *W3C Working Group Note*, . URL: http://www.w3.org/TR/wsdl.

Cicconi, F., 2005, 'Webstandards', URL: http://ruf.rockgrafia.com/webstandards/. Última Consulta: 10/2006.

Costello, R. L., 2007, 'Building web services the rest way', URL: http://www.xfront.com/REST-Web-Services.html. Última Consulta: 11/2007.

Cunha, D., 2002. 'Web services, soap e aplicações web'. *Netscape Devedge*, . URL: http://devedge-temp.mozilla.org/viewsource/2002/soap-overview/index_pt_br.html.

Erl, T., 2005. 'Introdução às tecnologias web services: Soa, soap, wsdl e uddi - parte 2'. *Revista Web Mobile*, (2), pp. 22–28.

Fallside, D. C. e Walmsley, P., 2004. 'Xml schema part 0: Primer second edition'. *W3C Recommendation*, . URL: `http://www.w3.org/TR/xmlschema-0/`.

Fonseca, F. T., 2004. Elementos finitos para modelos estruturais de barras. Dissertação de Mestrado, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil.

Fonseca, F. T. e Pitangueira, R. L., 2007. 'An object oriented class organization for dynamic geometrically non-linear fem analysis'. *CILAMCE 2007*, .

Fonseca, M. T., 2006. Aplicação orientada a objetos para análise fisicamente não-linear com modelos reticulados de seções transversais compostas. Dissertação de Mestrado, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil.

Fuina, J. S., 2006, 'Modelagem de meios parcialmente frágeis heterogêneos utilizando o contínuo de cosserat e o modelo de microplanos (projeto de tese de doutorado)', Universidade Federal de Minas Gerais. Belo Horizonte, MG, Brasil.

Gonçalves, M. A. B., 2004. Geração de malhas bidimensionais de elementos finitos baseada em mapeamentos transfinitos. Dissertação de Mestrado, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil.

Google, 2007, 'Portal google', URL: `http://www.google.com`. Última Consulta: 07/2007.

Hat, R. I., 2007, 'The fedora project', URL: `http://fedoraproject.org/`. Última Consulta: 08/2007.

Kurniawan, B. e Deck, P., 2004. *How Tomcat Works: A Guide to Developing You Own Java Servlet Container*.

Lozano, F., 2004. 'Aplicações web no tomcat 5 - parte 1: Primeiros passos no desenvolvimento de jsp'. *Revista Java Magazine*, vol. sIII(18), pp. 20–31.

Maven, A. S. F., 2007, 'Apache maven project', URL: `http://maven.apache.org/`. Última Consulta: 08/2007.

Moreira, R. N., 2004. Sistema gráfico interativo para ensino do método de elementos finitos. Dissertação de Mestrado, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil.

Nascimento, H. P., 2005. 'Modelos de implementação para aplicações web services clientes'. *Revista Mundo Java*, (11), pp. 24–30.

Penna, S. S., 2007. Pós-processador para modelos bidimensionais não-lineares do método dos elementos finitos. Dissertação de Mestrado, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil.

Ray, E. T., 2001. *Aprendendo XML*.

Saliba, S., 2007. Implementação computacional e análise crítica de elementos finitos de placas. Dissertação de Mestrado, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil.

Silva, M. S., 2006, 'Css para webdesign', URL: `http://www.maujor.com/tutorial/intrtut.php`. Última Consulta: 10/2006.

Smart, J. F., 2006. 'Rapid java web application development with tapestry'. *DevX.com*, . URL: `http://www.devx.com/Java/Article/30316/0/page/3`.

Sumra, R. e Arulazi, D., 2007, 'Quality of service for web services - demystification, limitations, and best practices', URL: `http://www.developer.com/services/article.php/10928_2027911_1`. Última Consulta: 07/2007.

Sun, M., 2007, 'Java.sun.com', URL: `http://java.sun.com/`. Última Consulta: 08/2007.

Systinet, C., 2005, 'How web services work', URL: `http://www.systinet.com/doc/ssc-65/primer/html/how.web.services.work.html`. Última Consulta: 07/2007.

Tapestry, A., 2006, 'Tapestry user's guide', URL: `http://tapestry.apache.org/tapestry4.1/usersguide/index.html`. Última Consulta: 10/2006.

Tomcat, A. S. F., 2007, 'Apache tomcat', URL: `http://tomcat.apache.org/`. Última Consulta: 08/2007.

Tong, K. I. K., 2005. *Enjoying Web Development with Tapestry*. TipTec Development. Book Website: `http://www.agileskills2.org`.

w3Schools, 2007*a*, 'Xsl-fo tutorial', URL: `http://www.w3schools.com/xslfo/default.asp`. Última Consulta: 07/2007.

w3Schools, 2007*b*, 'Xslt tutorial', URL: `http://www.w3schools.com/xsl/default.asp`. Última Consulta: 07/2007.