

FCT – Faculdade de Ciências e Tecnologia
DMEC – Departamento de Matemática, Estatística e Computação
Bacharelado em Ciência da Computação

Ricardo Kazuyuki Tanaka

Web Services Aplicados à Visualização
Monografia para Conclusão de Curso

Orientador: Prof. Dr. Milton Hirokazu Shimabukuro

RICARDO KAZUYUKI TANAKA

Web Services Aplicados à Visualização

Monografia apresentada ao Departamento de Matemática, Estatística e Computação (DMEC), da Faculdade de Ciência e Tecnologia (FCT), UNESP, como parte das atividades da disciplina Trabalho de Conclusão de Curso, necessária para a para obtenção do título de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Milton Hirokazu Shimabukuro

**Presidente Prudente
2010**

TERMO DE APROVAÇÃO

RICARDO KAZUYUKI TANAKA

Web Services aplicados à visualização

Monografia aprovada como requisito para obtenção do título de Bacharel em Ciência da Computação, da Faculdade de Ciências e Tecnologia Júlio de Mesquita Filho, FCT-UNESP, pela seguinte banca examinadora:

Orientador: Prof. Dr. Milton Hirokazu Shimabukuro
Departamento de Estatística, Matemática e Computação - UNESP

Prof. Dr. Marco Antônio Piteri
Departamento de Estatística, Matemática e Computação - UNESP

Prof. Dr. Almir Artero Olivetti
Departamento de Estatística, Matemática e Computação - UNESP

**Presidente Prudente
2010**

Dedicatória

Primeiramente aos meus pais e meu irmão, que sempre me apoiaram nos melhores e piores momentos da vida, e também me propiciaram toda a oportunidade e experiência vivenciada na faculdade. Mesmo em momentos de dificuldades nunca demonstraram fraqueza e sempre se esforçaram e se sacrificaram para que eu nunca desistisse. Dedico este trabalho também aos meus amigos que compartilharam minha estadia em Presidente Prudente - SP, amigos os quais levarei para sempre. Companheiros que sempre estiveram ao meu lado, proporcionando momentos alegres e agradáveis nesta fase tão importante e difícil da vida de cada um de nós.

Agradecimentos

Agradeço principalmente aos meus pais, aos quais devo tudo o que consegui até hoje. Nenhuma frase, nem dedicatória e agradecimentos desta monografia ou de qualquer outra que eventualmente possa escrever são suficientes para expressar a gratidão que devo aos meus pais. Agradeço também ao Prof. Dr. Milton Hirokazu Shimabukuro, o qual me auxiliou em todo este projeto, me mostrando os caminhos a percorrer e que também me apoiou e ajudou em uma mudança repentina em minha vida. Meus amigos Everton Toiço e Lucas Bolinha, os quais dividi moradia por cinco anos, Fernando Codorna, Allan, Carlos Moller, Diogo e Ademar, que também compartilharam lar nesses últimos anos de faculdade, além dos amigos Rafael, Felipe, Caio, Gustavo, Pedro e Álvaro, e das amigas Fernanda, Paula, Ágatha, Mayra, Caroline e Gabriela os quais considero amigos de hoje e sempre. Deixo aqui minha mensagem de gratidão e votos de sucesso. Sempre que precisarem podem contar comigo com o que for preciso. Muito obrigado a todos vocês que fizeram meus dias em Presidente Prudente - SP mais felizes e agradáveis.

Resumo

Com o crescimento acelerado do uso de aplicações *Web* em diversas áreas de conhecimento, o termo *Web service* entra em evidência no cenário atual, o qual resumidamente, refere-se a serviços das mais diferentes origens e objetivos, oferecidos através de redes locais e disponibilizados também, em alguns casos, na Internet. A arquitetura de tal tipo de aplicação propõe o processamento de dados do lado servidor, chamando a atenção, dessa forma, para a execução de aplicações e processos complexos e demorados, que é o caso da maioria dos algoritmos que envolvem visualização. A VTK é uma biblioteca destinada à visualização, e possui uma grande variedade de métodos e algoritmos para esta finalidade, porém com um motor gráfico que, naturalmente, requer capacidade de processamento. A união dos dois recursos pode trazer resultados interessantes e contribuir para a melhoria no desempenho para uso da biblioteca VTK. Tal estudo é abordado neste projeto, através de testes e análises de impacto na comunicação.

Palavras-chave: Visualização, *Web service*, *Internet*.

Abstract

With the rapid growth of the use of Web applications in various fields of knowledge, the term Web service enter into evidence in the current scenario, which refers to services from different origins and purpose, offered through local networks and also available in some cases, on the Internet. The architecture of this type of application offers data processing on server side thereby, running applications and complex and slow processes is very interesting, which is the case with most algorithms involving visualization. The VTK is a library intended for visualization, and features a large variety of methods and algorithms for this purpose, but with a graphics engine that requires processing capacity. The union of these two resources can bring interesting results and contribute for performance improvements in the VTK library. This study is discussed in this project, through testing and communication overhead analysis.

Keywords: *Visualization, Web service, Internet.*

Sumário

1	Introdução	1
1.1	Objetivos do Trabalho	1
1.2	Organização do Trabalho	2
2	Fundamentação	3
2.1	Web Service	5
2.1.1	Web Service Description Language (WSDL)	6
2.1.2	Simple Object Access Protocol (SOAP)	8
2.1.3	Universal Description, Discovery, and Integration (UDDI)	10
2.2	The Visualization Toolkit (VTK)	11
2.2.1	<i>Pipeline</i> de Visualização	12
2.2.2	Formato de arquivos VTK	15
3	Implementação	22
3.1	Web Services em JAVA/NetBeans	23
3.1.1	Caracterização dos Web Services	23
3.1.2	Troca de Mensagens	25
3.1.3	Consumo do serviço e WSDL	27
3.1.4	Utilização das classes VTK	27
3.1.5	Conversão de arquivos do tipo VRML para arquivos do tipo X3D	32
3.2	Repositório UDDI	33
3.2.1	Configuração e implantação do repositório Apache jUDDI	33
3.3	Cliente para o consumo dos serviços	34
3.3.1	Geração das classes de consumo na aplicação cliente	35
3.3.2	Envio dos arquivos do tipo VTK	38
3.3.3	Interface gráfica da aplicação	39
4	Impacto na comunicação: testes e resultados	41
4.1	Testes com PolyData	42
4.2	Testes com StructuredPoints	45
4.3	Análise dos resultados	48
5	Considerações finais	50
A	Configuração do servidor de aplicações Apache Tomcat 5.5	A-1
B	Criação de <i>Web services</i> utilizando a IDE NetBeans 6.9.1	B-1
C	Criação de um cliente para consumo de um <i>Web service</i> utilizando a IDE NetBeans 6.9.1	C-1

D	WSDL do <i>Web service</i> PolyDataCreator	D-1
E	WSDL do <i>Web service</i> UnstructuredGridCreator	E-1
F	WSDL do <i>Web service</i> StructuredPointsCreator	F-1
G	WSDL do <i>Web service</i> VRMLConverter	G-1

Lista de Figuras

2.1	Estrutura de um WSDL.	8
2.2	Estrutura de um <i>envelope</i> SOAP	9
2.3	Uso do registro UDDI.	11
2.4	Malha renderizada e Ray Cast em VTK.	12
2.5	Diagramas de classes: <i>vtkRenderWindow</i> , <i>vtkRenderer</i> e <i>vtkActor</i>	13
2.6	Exemplo de Rendering.	15
2.7	Cubo gerado através do arquivo VTK.	21
3.1	Troca de Mensagens.	26
3.2	Fluxo do processo de geração de um arquivo do tipo VRML a partir de um arquivo do tipo VTK.	30
3.3	Conversão do padrão VRML para X3D.	32
3.4	Pacotes e classes gerados pelo cliente para consumo dos <i>Web Services PolyDataCreator</i> e <i>UnstructuredGridCreator</i>	37
3.5	Pacotes e classes gerados pelo cliente para consumo dos <i>Web Services StructuredPointsCreator</i> e <i>VRMLConverter</i>	38
3.6	Interface gráfica da aplicação cliente.	40
4.1	Situação criada para realizar os testes.	41
4.2	Cena renderizada a partir do arquivo <i>fran_cut.vtk</i>	43
4.3	Resultado do primeiro dia de testes com o arquivo <i>fran_cut.vtk</i> . Os resultados são medidos em segundos	43
4.4	Gráficos dos resultados do primeiro dia de testes com o arquivo <i>fran_cut.vtk</i>	44
4.5	Resultado do segundo dia de testes com o arquivo <i>fran_cut.vtk</i> . Os resultados são medidos em segundos	44
4.6	Gráficos dos resultados do segundo dia de testes com o arquivo <i>fran_cut.vtk</i>	45
4.7	Cena renderizada a partir do arquivo <i>ironProt.vtk</i>	46
4.8	Resultado do primeiro dia de testes com o arquivo <i>ironProt.vtk</i> . Os resultados são medidos em segundos	46
4.9	Gráficos dos resultados do primeiro dia de testes com o arquivo <i>ironProt.vtk</i>	47
4.10	Resultado do segundo dia de testes com o arquivo <i>ironProt.vtk</i> . Os resultados são medidos em segundos	47
4.11	Gráficos dos resultados do segundo dia de testes com o arquivo <i>ironProt.vtk</i>	48
4.12	Tabela de análise dos testes com <i>PolyData</i>	49
4.13	Tabela de análise dos testes com <i>StructuredPoints</i>	49
A.1	Página inicial do Apache Tomcat 5.5.	A-1
A.2	Página da ferramenta administrativa do Apache Tomcat 5.5. . . .	A-2

A.3	Página do Tomcat Manager.	A-2
B.1	Criação de um projeto Web comum.	B-1
B.2	Local e nome do projeto Web.	B-2
B.3	Escolha do servidor e da versão do J2EE.	B-2
B.4	Criação de um arquivo do tipo Serviço Web (<i>Web Service</i>).	B-3
B.5	Nome e pacote onde será armazenado o arquivo <i>Web Service</i>	B-3
C.1	Nova classe para consumo de um <i>Web Service</i>	C-1
C.2	Indicação do WSDL do <i>Web service</i> para consumo.	C-2
C.3	Pacotes de classes gerados pela IDE NetBeans.	C-2

Capítulo 1

Introdução

Este projeto aborda a arquitetura utilizada por *Web services* em conjunto com a biblioteca gráfica VTK, bem como os elementos envolvidos nas implementações e troca de informações, tais como o protocolo SOAP e a linguagem WSDL. O projeto proporcionou um estudo do impacto da comunicação em rede no uso da biblioteca VTK ao utilizar processamento local e processamento não local, graças ao uso da arquitetura dos *Web services*.

1.1 Objetivos do Trabalho

O alvo principal deste projeto consiste em estudar a comunicação entre os *Web services* desenvolvidos e a aplicação cliente, analisando o impacto, transparência e integridade no envio e recebimento dos dados. Tal comunicação se dá através de protocolos e padrões já definidos e bastante utilizados no cenário atual.

Os *Web services* principais do projeto fazem uso de uma biblioteca gráfica, a VTK (*Visualization Toolkit*), a qual possui alto poder computacional, com algoritmos complexos e poderosos em relação à computação gráfica, porém envolve bastante processamento. É neste cenário que se aplica a razão em utilizar a arquitetura de comunicação dos *Web services*, concentrando a execução dos algoritmos gráficos do lado das máquinas servidoras, as quais geralmente possuem alto poder de processamento, e deixando a cargo da aplicação cliente apenas a visualização da cena já renderizada pelo *Web service*.

Uma característica bastante útil e interessante com relação aos *Web services* em geral, é que a comunicação com os mesmos se dá através de um protocolo conhecido como SOAP (*Simple Object Access Protocol*), o qual especifica um documento XML que contém as mensagens de troca de informações entre cliente e *Web service*, transportado através do protocolo HTTP ou FTP, possibilitando, assim a comunicação entre clientes e *Web services* desenvolvidos em qualquer plataforma que possua suporte a esse tipo de arquitetura e linguagem de programação, ficando assim independente, portátil e operacional a relação entre as aplicações.

1.2 Organização do Trabalho

O Capítulo 2 contém a fundamentação teórica, o qual se discute a fundo cada componente utilizado neste projeto. Documentado a partir de estudos e pesquisas realizadas ao longo do projeto, necessários para a correta implementação do mesmo.

O Capítulo 3 destina-se aos detalhes de implementação de cada módulo do projeto, como foram configurados, desenvolvidos e utilizados no escopo das aplicações.

O Capítulo 4 discute sobre os testes de impacto na comunicação pela rede realizados sobre as aplicações construídas, com análises de desempenho e comparativos exibidos graficamente.

Por fim, no Capítulo 5 é feita uma conclusão dos prós e contras ao utilizar uma arquitetura baseada em *Web services*, os benefícios alcançados, dificuldades apresentadas e considerações finais do autor, apresentando possibilidades de futuros trabalhos e pesquisas na área deste projeto.

Capítulo 2

Fundamentação

Web Services são serviços disponibilizados na *Internet* que utilizam documentos escritos na linguagem XML (*eXtensible Markup Language*) como padrão de comunicação entre as aplicações que solicitam seu serviço e um protocolo de transferência de dados via *Internet* para que seja possível a comunicação entre solicitante e fornecedor de serviços.

Segundo [CHAPPELL e JEWELL \(2002\)](#), um *Web Service* é uma fatia da lógica de negócios, localizado em algum lugar na *Internet*, que é acessível através de protocolos de *Internet* baseados em padrões, tais como HTTP ou SMTP. Usar o serviço de um *Web Service* pode ser tão simples como fazer *login* em um site ou tão complexo como facilitar uma negociação entre múltiplas organizações empresariais.

A organização W3C¹, a qual estabelece normas e padrões aos *Web Services* define no documento de [FERRIS et al. \(2004\)](#) *Web Service* como um sistema de software projetado para suportar interação máquina-máquina interoperáveis sobre uma rede. Tem uma interface descrita em um formato processável por máquina (especificamente WSDL). Outros sistemas interagem com o *Web Service* de uma maneira prescrita por sua descrição usando mensagens SOAP, tipicamente transmitida usando HTTP com serialização XML em conjunto com outras normas relacionadas à Web.

Uma característica bastante interessante e importante envolvendo o conceito de *Web Service* é a interoperabilidade entre as aplicações, não dependendo da linguagem, plataforma ou sistema operacional que estas foram concebidas.

Para [CERAMI \(2006\)](#), um *Web Service* é qualquer serviço que está disponível na *Internet*, usa a linguagem XML como padrão de troca de mensagens, e não está vinculada a qualquer sistema operacional ou uma linguagem de programação.

Além de utilizar a linguagem XML e um protocolo de transferência de *Internet* (no caso deste projeto HTTP), de acordo com [JOSUTTIS \(2007\)](#) existem ainda mais 3 conceitos fundamentais a respeito de *Web Service*. Tais conceitos são específicos a *Web Service* e foram os primeiros padrões para tal:

- **WSDL** é usado para definir as interfaces de serviço. Pode descrever dois aspectos diferentes de um serviço: sua assinatura (nome e parâmetros) e detalhes de seus *bindings* e de seu desenvolvimento (protocolo e localização);

¹ sigla para World Wide Web Consortium, entidade que visa desenvolver padrões para a Web

- **SOAP** é o padrão que define o protocolo dos *Web Services*. Especifica o formato para a troca de dados entre *Web Services* sobre o protocolo HTTP;
- **UDDI** é o padrão de gerenciamento dos *Web Services* (registrar e procurar serviços, por exemplo).

CHAPPELL e JEWELL (2002) afirmam ainda que *Web Services* possuem algumas características comportamentais especiais:

- **Baseado em XML**

Ao utilizar XML como representação de dados para todos os protocolos e tecnologias acerca de *Web Services*, essas tecnologias podem ser interoperáveis em baixo-nível. Como transporte de dados, XML elimina as barreiras dos protocolos de qualquer rede, sistema operacional, ou plataforma;

- **Baixo Acoplamento**

Um sistema com alto acoplamento implica que o cliente e o servidor são fortemente ligados uns aos outros, ou seja, ao ocorrer mudanças de interface em um, o outro também deverá ser atualizado. A adoção de uma arquitetura de baixo acoplamento tende a tornar os sistemas mais gerenciáveis e permite a integração entre diferentes sistemas de maneira simples;

- **Alta Coesão**

Linguagens orientadas a objeto, como Java por exemplo, expõem seus serviços através de métodos. Um método individual é pouco coeso, se tratando em fornecer qualquer operação útil em nível empresarial. Construir um programa contendo vários métodos compõem então um serviço altamente coeso que é consumido por um cliente ou outro serviço. Os negócios e as interfaces que eles expõem devem possuir alta coesão. Os *Web Services* fornecem uma maneira natural de definir serviços altamente coesos;

- **Capacidade em ser Síncrono ou Assíncrono**

A sincronicidade refere-se a ligação entre o cliente e a execução do serviço. Em invocações síncronas, o cliente interrompe sua execução e aguarda a resposta do serviço para completar seu fluxo de execução. Operações assíncronas permitem a um cliente invocar um serviço e executar outras funções. Clientes assíncronos recuperam seu resultado em um ponto mais tarde, enquanto que os clientes síncronos recebem seu resultado assim que o serviço foi concluído. A capacidade de um *Web Service* ser assíncrono é um fator essencial para permitir baixo acoplamento entre os sistemas;

- **Suporte à Chamadas Remotas de Procedimentos (RPCs)**

Web services permitem que os clientes invoquem procedimentos, funções ou métodos remotamente usando um protocolo baseado em XML. Procedimentos remotos expõem os parâmetros de entrada e saída que um *Web Service* suporta;

- **Suporte a Troca de Documentos**

Uma das principais vantagens do XML é a sua maneira genérica de representar não somente dados, mas também documentos complexos. Estes documentos podem ser simples, como quando representando um endereço físico qualquer, ou pode ser complexo, representando um livro inteiro, por exemplo. *Web Services* suportam a troca transparente de documentos para facilitar a integração dos negócios.

Introduzido os padrões e normas que cercam o conceito de *Web Service*, este trabalho também provê o uso de uma ferramenta bastante poderosa em termos de processamento e manipulação de imagens, a biblioteca gráfica VTK.

Em [vtk \(2010\)](#) segue a definição de que o Visualization Toolkit (VTK) é um software de código-fonte aberto, disponível gratuitamente para computação gráfica 3D, processamento e visualização de imagem utilizado por milhares de pesquisadores e desenvolvedores ao redor do mundo. VTK consiste em uma biblioteca de classes C++ e várias camadas de interpretação, incluindo interface Tcl / Tk, Java e Python.

A grande vantagem em se utilizar tal ferramenta neste projeto se dá em função das inúmeras funções e algoritmos complexos já implementados, como se pode observar em [vtk \(2010\)](#), VTK suporta uma grande variedade de algoritmos de visualização, incluindo métodos escalares, vetores, tensores, de textura e volumétricos e técnicas avançadas de modelagem, tais como modelagem implícita, redução de polígonos, malhas de alisamento, corte de contorno e triangulação de Delaunay.

2.1 Web Service

A definição dada a *Web Service* por [PULIER e TAYLOR \(2002\)](#) é que um *Web Service* é uma peça de software que obedece a um conjunto de padrões não proprietários de interoperabilidade. Estes padrões permitem a interação global das aplicações, independentemente da plataforma de hardware, sistema operacional, infra-estrutura de rede, ou linguagem de programação. De fato, tal definição é extremamente racional, visto que um dos propósitos quando se utiliza *Web Services* é justamente essa interoperabilidade e independência entre os sistemas. [PULIER e TAYLOR \(2002\)](#) ainda dizem que a extraordinária utilidade do *Web Service* é baseada no fato de que ele usa protocolos abertos de comunicação na Internet e XML para as suas atividades. Um *Web Service* é, portanto, uma parte do software que pode agir a pedido de qualquer aplicação conectada à rede no mundo que se comunica através de seus padrões.

Para que todas estas características atribuídas ao conceito de *Web Service* funcionem do modo adequado e coerente, deve-se conhecer a fundo os padrões e normas aplicados ao mesmo. As próximas Seções deste Capítulo, são destinadas com este propósito, explorar as principais tecnologias que compõem a essência de *Web Service*, segundo [CHAPPELL e JEWELL \(2002\)](#) e também [JOSUTTIS \(2007\)](#), Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP) e Universal Description, Discovery, and Integration (UDDI).

2.1.1 Web Service Description Language (WSDL)

De acordo com [BARRY \(2003\)](#) o WSDL forma a base para os *Web Services*, portanto este é um conceito de extrema relevância diante do tema deste projeto.

Basicamente, o WSDL é um documento escrito na linguagem XML, seguindo os padrões especificados pela W3C, que descreve como e o que um *Web Service* tem a oferecer.

[CHAPPELL e JEWELL \(2002\)](#) resumem um documento WSDL como uma receita usada para automatizar os detalhes envolvidos na comunicação entre aplicações. [CHAPPELL e JEWELL \(2002\)](#) dizem ainda que, tudo o que é definido dentro de um arquivo WSDL é abstrato, são apenas definições de parâmetros e restrições sobre como a comunicação deve ocorrer durante a execução. O arquivo WSDL deverá conter as diretrizes definidas através da implementação de um *Web Service*.

Quanto à estrutura de um arquivo WSDL, o documento de [CHRISTENSEN et al. \(2001\)](#)² especifica os seguintes elementos básicos para o uso nas definições de um serviço:

- **Types** um *container* para as definições de tipo de dados usando algum tipo de sistema (como XSD);
- **Message** uma definição abstrata dos tipos de dados que são comunicados;
- **Operation** uma definição abstrata de uma ação suportada pelo serviço;
- **Port Type** um conjunto de uma ou mais operações suportadas pelo serviço. Cada operação se refere a uma mensagem de entrada ou de saída;
- **Binding** um protocolo concreto e uma especificação de dados para um tipo de porta em particular;
- **Port** um terminal único definido como uma combinação de um *binding* e um endereço de rede;
- **Service** uma coleção de terminais relacionados.

Existem outros elementos em um WSDL, no entanto os elementos citados acima, possuem uma relevância maior ao se descrever um *Web Service* através de um documento WSDL.

Abaixo segue um exemplo retirado de [CHAPPELL e JEWELL \(2002\)](#) mostrando a estrutura básica de um documento WSDL, representado também pela Figura 2.1.

²Disponível em <http://www.w3.org/TR/wsdl#A4.4>

```
<definitions>
  <import>
  <types>
    <schema></schema>
  </types>
  <message>
    <part></part>
  </message>
  <PortType>
    <operation>
      <input></input>
      <output></output>
      <fault></fault>
    </operation>
  </PortType>
  <binding>
    <operation>
      <input></input>
      <output></output>
    </operation>
  </binding>
  <service>
    <port></port>
  </service>
</definitions>
```

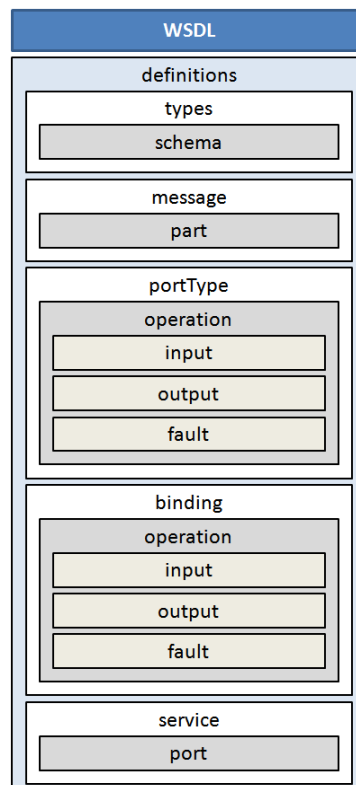


Figura 2.1: Estrutura de um WSDL.

Devido ao WSDL, os *Web Services* são conhecidos como "auto-descritores". Este é um conceito muito poderoso, pois dessa forma os *Web Services* podem ser descritos universalmente, diz [PULIER e TAYLOR \(2002\)](#). Assim, um programador no Brasil pode criar um software que faz uso de um serviço oferecido por um *Web Service* implementado por uma pessoa no Japão, por exemplo, apenas lendo e processando o WSDL do respectivo *Web Service*.

2.1.2 Simple Object Access Protocol (SOAP)

Uma descrição bem sucinta e simples de SOAP é a citada por [MUELLER \(2002\)](#), o qual diz que SOAP é um protocolo de comunicação baseado em XML. Ele permite que aplicativos e componentes troquem dados.

SOAP comumente utiliza HTTP como o protocolo de transporte, no entanto pode fazer uso de qualquer outro protocolo para a transferência de dados. Tomaremos o HTTP como protocolo de transporte para este projeto.

Originalmente, o protocolo SOAP foi implementado pela *Microsoft*, porém ao contrário de outros produtos da mesma empresa, este é um protocolo livre e independente de sistema operacional.

Segundo [MUELLER \(2002\)](#), três características tornam SOAP atraente:

- Não existe vínculo com nenhum componente tecnológico - SOAP teoricamente funciona em qualquer plataforma;
- Não existe vínculo com nenhuma linguagem de programação específica - SOAP pode ser usado em qualquer linguagem;
- Fácil de aprender e simples de estender.

Como dito anteriormente, o protocolo SOAP utiliza a linguagem XML, assim como o documento WSDL (discutido na Seção anterior). Pela definição de ENGLANDER (2002), todas as mensagens do protocolo, são empacotadas em um documento XML chamado *envelope*. A metáfora é apropriada porque junta-se tudo o que é necessário para se realizar uma operação em um "envelope", o envia para um cliente que abre o "envelope" e reconstrói o conteúdo original, permitindo assim a realização da operação requisitada.

A estrutura básica de um *envelope* (ENGLANDER, 2002) deve conter o elemento *body* e pode conter um elemento opcional *header*, caso exista este elemento *header*, o mesmo deve ser o primeiro subelemento do *envelope* e então o elemento *body* deve aparecer logo após o *header*. Como ilustra a Figura 2.2.

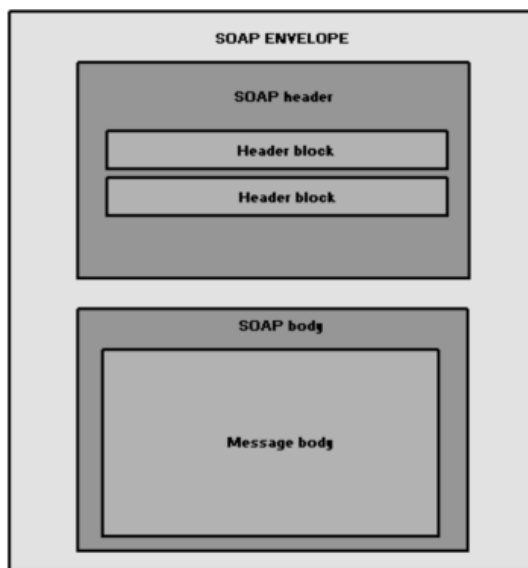


Figura 2.2: Estrutura de um *envelope* SOAP
(fonte:

<http://tech.konnectingtheworld.com/2010/10/an-overview-of-soap-protocol/>).

O trecho a seguir, adaptado de CHAPPELL e JEWELL (2002) mostra a estrutura básica de um arquivo XML encapsulado por um *envelope* conforme o protocolo SOAP.

```
<?xml version='1.0' encoding='UTF-8' ?>
<SOAP-ENV:Envelope
  ...
  <SOAP-ENV:Header>
    ...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Geralmente, o *header* do *envelope* contém as entradas usadas para codificar os elementos usados para o processo de transação, autenticação ou outras

informações relacionadas ao processamento ou encaminhamento da mensagem. Lembrando apenas que o *header* é um elemento opcional no *envelope*.

Dentro do elemento *body* encontramos a solicitação (*request*) ou resposta (*response*) de um serviço. Neste elemento, pode-se encontrar uma mensagem que contém o nome de um método e seus parâmetros, ou uma mensagem contendo suas partes relevantes, ou uma falha e seus detalhes (ENGLANDER, 2002).

2.1.3 Universal Description, Discovery, and Integration (UDDI)

Segundo CERAMI (2006), UDDI é uma especificação técnica para descrever, descobrir e integrar *Web Services*. Esta breve definição pode ser complementada pelo comentário encontrado em (BARRY, 2003), o qual diz que o registro UDDI pode ser pesquisado de várias formas para se obter informações de contato e *Web Services* disponíveis. CHAPPELL e JEWELL (2002) colocam que o projeto UDDI é uma iniciativa da indústria que tenta criar uma plataforma independente, *open framework* para descrever serviços, descobrir negócios e integrar serviços.

Embora vários tipos de registros de *Web Services* estejam disponíveis para uso, o UDDI é identificado como o diretório geral padrão, uma espécie de "páginas amarelas" dos *Web Services*, de acordo com PULIER e TAYLOR (2002).

Assim como SOAP e WSDL, a especificação UDDI utiliza a linguagem XML, e é dividida em três seções:

- **White Pages** Contêm informações básicas sobre a empresa, tal como nome, endereço, informações de contato e um identificador único. Estas informações permitem a descoberta de *Web Services* baseados nas identificações da empresa;
- **Yellow Pages** Incluem informações, dados gerais de classificação para cada empresa ou os tipos de serviços oferecidos por ela. Estas informações permitem a descoberta de *Web Services* de acordo com sua taxonomia;
- **Green Pages** Descrevem informações técnicas de comportamento e funções específicas que um *Web Service* possui. Geralmente, possuem o endereço (URL) onde os serviços do *Web Service* podem ser invocados.

A figura 2.3 representa genericamente o uso de um servidor UDDI.

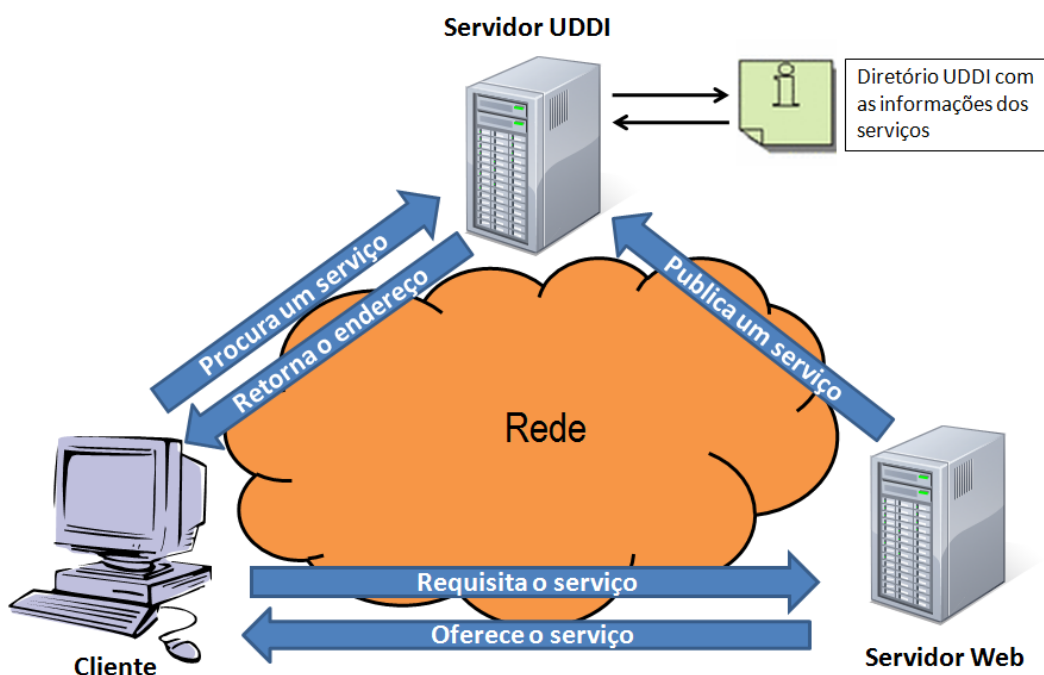


Figura 2.3: Uso do registro UDDI.

Analisados e compreendidos todos os componentes básicos e essenciais para o perfeito funcionamento e implementação de *Web services*, bem como a arquitetura de comunicação e transporte dos dados, a próxima Seção é destinada ao estudo da biblioteca VTK, seus processos de geração de imagens e a estrutura dos arquivos de dados utilizados pela biblioteca.

2.2 The Visualization Toolkit (VTK)

Visualization Toolkit (VTK) é uma biblioteca *open source* destinada à computação gráfica 2D/3D, processamento e visualização de imagens desenvolvida na linguagem C++ e disponível para utilização em outras linguagens, dentre elas JAVA e Python. Segundo consta em [vtk \(2010\)](#), é uma ferramenta utilizada por milhares de pesquisadores e desenvolvedores em todo o mundo. Apesar de se tratar de uma poderosa ferramenta no que se diz respeito à processamento e manipulação de imagens seu motor gráfico, necessita de um ambiente computacional de alto desempenho.

Para compensar essa perda de desempenho, assim como no trabalho de [SHU e CHEN \(2008\)](#), o uso da biblioteca VTK juntamente com Web Services se torna algo extremamente atraente, pois o processamento gráfico pode ser feito não-localmente ou até mesmo distribuído, ficando a cargo local apenas a visualização da imagem.

A Figura 2.4 mostra o poder de processamento gráfico da ferramenta.

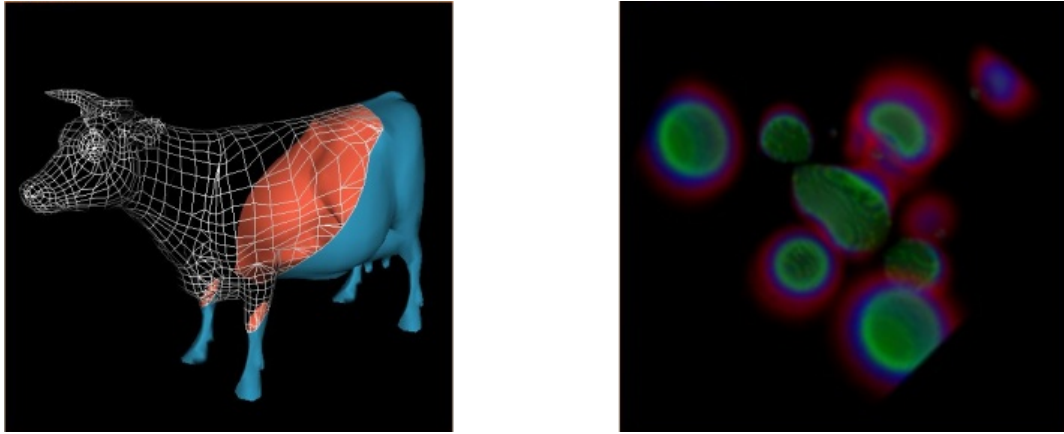


Figura 2.4: Malha renderizada e Ray Cast em VTK.

2.2.1 Pipeline de Visualização

Para se apresentar objetos (*rendering*) em computação gráfica é necessário definir uma cena. Em VTK a cena é definida, criando uma *RenderWindow* e um *Renderer*. Objetos, chamados atores (*Actors*) são definidos e adicionados à cena, que é mapeada e apresentada quando o programador desejar (SCHROEDER e MARTIN, 2005). Para tal processo, dá-se o nome de pipeline de renderização (*rendering pipeline*). De forma geral, o fluxo de execução para a visualização em VTK segue os seguintes passos:

- Criar os dados a serem visualizados;
- Filtrar os dados para as técnicas desejadas;
- Criar uma janela de *rendering*;
- Criar um *renderer*;
- Mapear os dados;
- "Renderizar" a cena.

A Figura 2.5 representa os diagramas de classes de herança dos elementos citados acima.

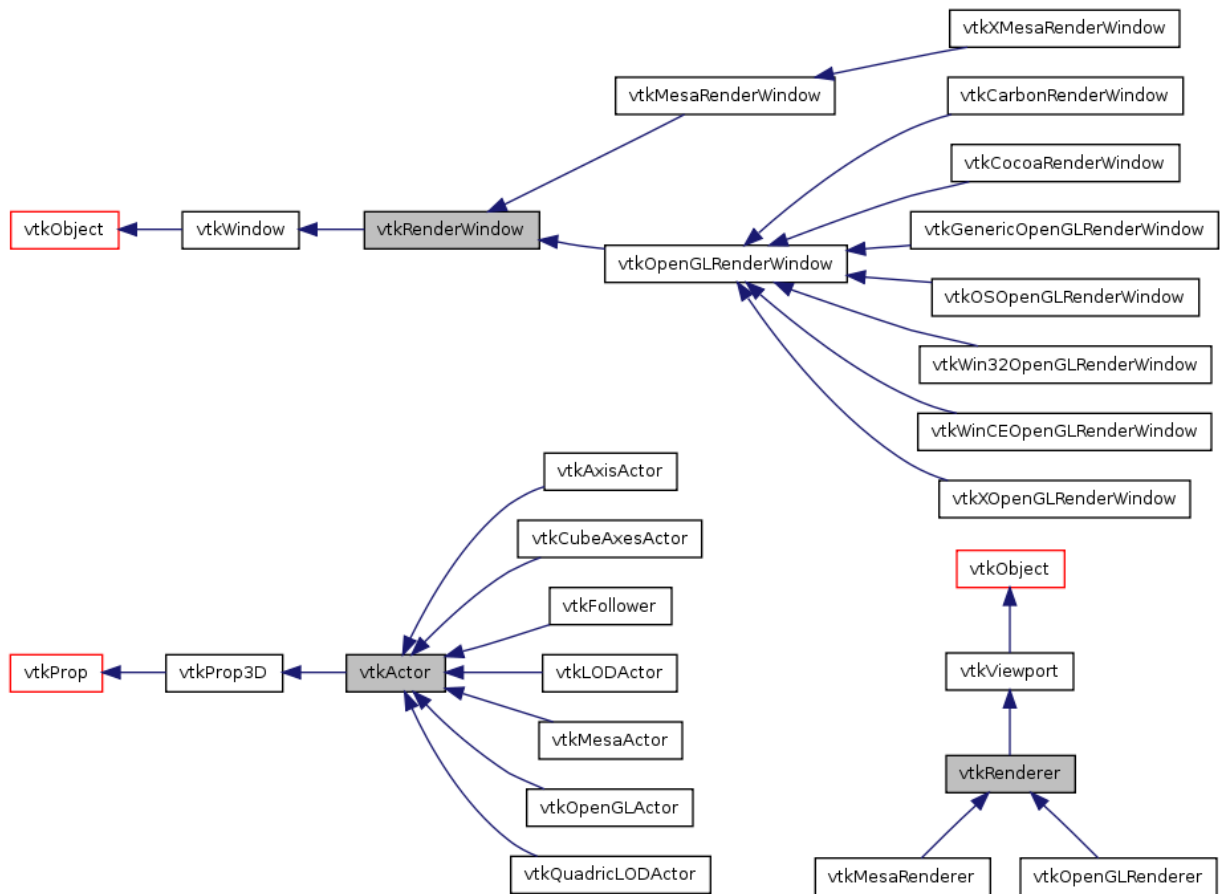


Figura 2.5: Diagramas de classes: `vtkRenderWindow`, `vtkRenderer` e `vtkActor`.
(fonte: <http://www.vtk.org>)

Como dito anteriormente, a VTK é uma biblioteca com fins gráficos escrita na linguagem C++, portanto trata-se de uma biblioteca de classes, ou seja, utiliza o paradigma orientado a objetos. Em razão disso, o pipeline de visualização (*visualization pipeline*) consiste em instanciar, caracterizar e adicionar objetos, definidos pelas classes da VTK, em uma cena.

Segundo [SCHROEDER e MARTIN \(2005\)](#), a visualização é inerentemente um processo de transformação: os dados são repetidamente transformados por uma sequência de operações de filtragem para produzir imagens. O papel do pipeline de visualização é de transformar informação em dados gráficos. Outra maneira de olhar para isto é que o pipeline de visualização é responsável por construir a representação geométrica.

A fim de melhor compreender o processo de formação e visualização de imagens através da VTK, faz-se necessário distinguir dois conceitos estruturais fundamentais: topologia e geometria.

- **Topologia** é o conjunto de informações que caracterizam o tipo de objeto de visualização, tal como a forma e a quantidade de lados de um polígono, ou uma superfície;
- **Geometria** é o conjunto de dados que caracterizam a dimensão e localização espacial do objeto na cena, como a localização dos pontos que formam um polígono no plano cartesiano.

Tomando um triângulo como exemplo, o mesmo pode ser, *isóceles*, *escaleno* ou *equilátero*, no entanto, para qualquer dos tipos a topologia é a mesma, ou seja, possuem 3 lados. Entretanto, se diferem em relação à geometria, pois cada um dos tipos possui uma peculiaridade quanto à medida espacial dos lados que o formam.

O *pipeline* de visualização e renderização para um objeto em VTK pode ser codificado conforme o trecho abaixo:

```
// instanciação e caracterização (geometria) de um
// objeto do tipo vtkConeSource
vtkConeSource coneSource = new vtkConeSource();
coneSource.SetRadius(1);
coneSource.SetAngle(15);
coneSource.SetHeight(0.5);
coneSource.SetResolution(50);

// mapeamento dos dados a serem apresentados na tela
vtkPolyDataMapper coneMapper = new vtkPolyDataMapper();
coneMapper.SetInput(coneSource.GetOutput());

// instanciação de um renderer
vtkRenderer renderer = new vtkRenderer();

// instanciação de um renderWindow
vtkRenderWindow renderWindow = new vtkRenderWindow();
renderWindow.AddRenderer(renderer);

// adição do ator na cena
renderer.AddActor(coneActor);
```

Podemos observar o resultado do trecho de código acima na figura [2.6](#).

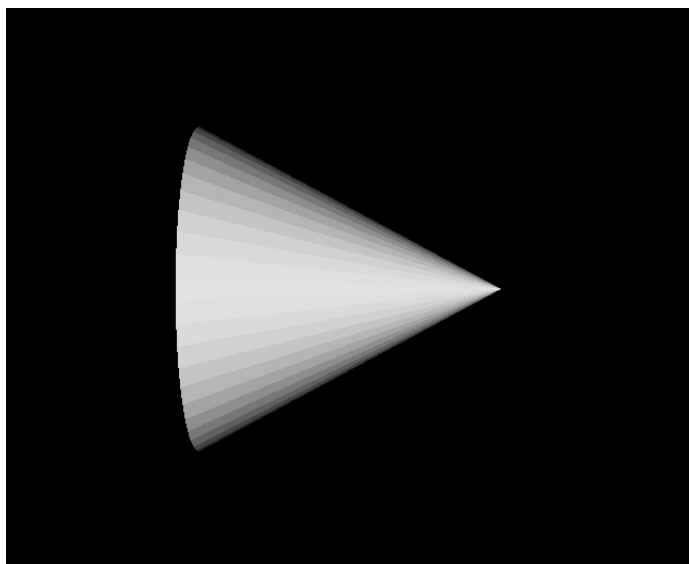


Figura 2.6: Exemplo de Rendering.

2.2.2 Formato de arquivos VTK

A biblioteca VTK ainda dispõe de classes capazes de ler e escrever diferentes tipos de dados comuns, como o caso de JPG, BMP e AVI, além disso provê um outro tipo de dado específico para a biblioteca. Existem duas maneiras de escrever um arquivo VTK, uma mais simples, de forma fácil de se escrever manualmente ou computacionalmente, outro mais flexível, baseado em XML. De acordo com [LAW \(2005\)](#) a estrutura de um arquivo VTK consiste em cinco partes:

1. A primeira parte é a versão do arquivo e o identificador. Esta parte contém simplesmente a linha: `# vtk DataFile Version x.x`. Esta linha deve ser exatamente como mostrada, com exceção do número da versão `x.x`, o qual irá variar de acordo com os diferentes releases³ da VTK;
2. A segunda parte é o cabeçalho. O cabeçalho consiste em uma string de caracteres terminada pelo caractere de final de linha `\n`. O cabeçalho pode conter no máximo 256 caracteres e pode ser utilizado para descrever o arquivo e incluir outras informações pertinentes;
3. A terceira parte descreve o formato do arquivo, que pode ser do tipo ASCII ou binário. As palavras-chave ASCII ou BINARY devem aparecer nessa linha;
4. A próxima parte é a estrutura do dataset, a qual descreve a geometria e a topologia da estrutura. Esta parte começa com uma linha contendo a palavra-chave DATASET seguida de outra palavra-chave que descreve o tipo do dataset;

³A versão atual é 3.0, arquivos das versões 1.0 e 2.0 são compatíveis com arquivos da versão 3.0

5. A quinta e última parte descreve os atributos do dataset. Esta parte começa com as palavras-chave POINT_DATA ou CELL_DATA, seguida por um número inteiro especificando o número de pontos ou células, respectivamente. Outras combinações de palavras-chave definem os valores dos atributos atuais do dataset (por exemplo, valores escalares, vetoriais, tensores, normais).

As primeiras três partes do arquivo são obrigatórias, as outras duas são opcionais. As palavras-chave não são *case sensitive*, e devem ser separadas por espaços em branco.

Abaixo segue sucintamente a estrutura de um arquivo⁴ VTK:

```
#vtkDataFileVersion2.0 } (1)
Reallycooldata } (2)
ASCII|BINARY } (3)
DATASETtype } (4)
...
POINT_DATA n } (5)
...
CELL_DATA n
...
```

Na quarta parte do arquivo (DATASET), a qual descreve a geometria e topologia da estrutura, um dos seguintes valores podem ser utilizados: STRUCTURED_POINTS, STRUCTURED_GRID, UNSTRUCTURED_GRID, POLYDATA, RECTILINEAR_GRID ou FIELD.

Abaixo, segue uma breve descrição de cada um dos itens citados acima, conforme [LAW \(2005\)](#)

- **STRUCTURED_POINTS:** Este formato descreve pontos estruturados em 1D, 2D ou 3D. As dimensões em x, y, z devem ser valores maiores ou igual a 1, e os valores de espaçamento em x, y, z devem ser maiores ou igual a zero.

```
DATASET STRUCTURED_POINTS
DIMENSIONS x y z
ORIGIN x y z
SPACING x y z
```

- **STRUCTURED_GRID:** Este formato descreve malhas estruturadas em 1D, 2D ou 3D. As dimensões em x, y, z devem ser valores maiores ou igual a 1. As coordenadas dos pontos são definidas na seção POINTS e consistem em valores x-y-z para cada ponto dado.

```
DATASET STRUCTURED_GRID
DIMENSIONS x y z
POINTS n dataType
p 0x p 0y p 0z
p 1x p 1y p 1z
...
p (n-1) x p (n-1) y p (n-1) z
```

⁴Retirado de [LAW \(2005\)](#)

- **UNSTRUCTURED_GRID:** Este tipo de formato consiste em combinações arbitrárias de cada tipo possível de célula. Malhas não estruturadas são definidas por pontos, células e tipos de células. A palavra-chave CELLS requer dois parâmetros: o número de células n e o tamanho da lista de células. O tamanho da lista de células é o número total de valores requeridos para representar a lista. A palavra-chave CELL_TYPE requer um simples parâmetro: o número de células n . Este valor deve coincidir com o valor especificado pela palavra-chave CELLS. O tipo de célula é um valor inteiro que especifica o tipo de cada célula.

```

DATASET UNSTRUCTURED_GRID
POINTS n dataType
p 0x p 0y p 0z
p 1x p 1y p 1z

...
p (n-1) x p (n-1) y p (n-1) z

CELLS n size
numPoints 0 ,i, j, k, l,...
numPoints 1 ,i, j, k, l,...
numPoints 2 ,i, j, k, l,...
...
numPoints n-1 ,i, j, k, l,...
Simple Legacy Formats 5
CELL_TYPER n
type 0
type 1
type 2
...
type n-1

```

- **POLYDATA:** O formato poligonal consiste em combinações arbitrárias de superfícies gráficas de vértices primitivos, linhas, polígonos e triângulos. Este formato é definido pelas seções POINTS, VERTICES, LINES, POLYGONS ou TRIANGLE_STRIPS. A definição da seção POINTS é a mesma que a citada anteriormente para o tipo UNSTRUCTURED_GRID. As palavras-chave VERTICES, LINES, POLYGONS ou TRIANGLE_STRIPS definem a topologia poligonal da estrutura. Cada uma dessas palavras-chave requerem dois parâmetros: o número de células n e o tamanho da lista de células, no entanto, nenhuma dessas últimas palavras-chave são obrigatórias em um arquivo do tipo POLYDATA.

```

DATASET POLYDATA
POINTS n dataType
p 0x p 0y p 0z
p 1x p 1y p 1z

...
p (n-1) x p (n-1) y p (n-1) z

VERTICES n size
numPoints 0 , i 0 , j 0 , k 0 , ...
numPoints 1 , i 1 , j 1 , k 1 , ...
...
numPoints n-1 , i n-1 , j n-1 , k n-1 ,
...

LINES n size
numPoints 0 , i 0 , j 0 , k 0 , ...
numPoints 1 , i 1 , j 1 , k 1 , ...
...
numPoints n-1 , i n-1 , j n-1 , k n-1 ,
...

POLYGONS n size
numPoints 0 , i 0 , j 0 , k 0 , ...
numPoints 1 , i 1 , j 1 , k 1 , ...
...
numPoints n-1 , i n-1 , j n-1 , k n-1 ,
...

TRIANGLE_STRIP n size
numPoints 0 , i 0 , j 0 , k 0 , ...
numPoints 1 , i 1 , j 1 , k 1 , ...
...
numPoints n-1 , i n-1 , j n-1 , k n-1 ,
...

```

- **RECTILINEAR_GRID:** Este tipo de formato é definido por uma topologia regular, uma geometria semi-regular alinhada através dos eixos x-y-z. A geometria é definida por três listas de valores das coordenadas, uma lista para cada eixo x-y-z. A topologia é definida especificando a malha de dimensões, a qual deve ser maior ou igual a 1.

```

DATASET RECTILINEAR_GRID
DIMENSIONS n x n y n z
X_COORDINATES n x dataType
x 0 x 1 ... x (nx-1)
Y_COORDINATES n y dataType
y 0 y 1 ... y (ny-1)
Z_COORDINATES n z dataType
z 0 z 1 ... z (nz-1)

```

- **FIELD:** O formato FIELD é um formato genérico sem estrutura topológica ou geométrica, e sem uma dimensão em particular. Tipicamente, este tipo de formato é associado a pontos ou células de um dataset. No entanto, se o tipo FIELD estiver especificado como um tipo de dataset, então um arquivo VTK genérico é definido.

Para os atributos do dataset, que estão definidos na quinta e última parte do arquivo VTK, são aceitos os seguintes tipos: SCALARS, VECTORS, NORMALS, TENSORS, TEXTURE_COORDINATES, FIELD e LOOKUP_TABLE.

Um exemplo de um arquivo VTK completo pode ser visualizado a seguir, retirado de [LAW \(2005\)](#):

```
# vtk DataFile Version 2.0
Cube example
ASCII
DATASET POLYDATA
POINTS 8 float
0.0 0.0 0.0
1.0 0.0 0.0
1.0 1.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
1.0 0.0 1.0
1.0 1.0 1.0
0.0 1.0 1.0
POLYGONS 6 30
40 1 2 3
44 5 6 7
40 1 5 4
42 3 7 6
40 4 7 3
41 2 6 5

CELL_DATA 6
SCALARS cell_scalars int 1
LOOKUP_TABLE default
0
1
2
3
4
5
```

```

NORMALS cell_normals float
8 VTK 4. 2 File Formats
0 0 -1
0 0 1
0 -1 0
0 1 0
-1 0 0
1 0 0
FIELD FieldData 2
cellIds 1 6 int
01 2 3 4 5
faceAttributes 2 6 float
0.0 1.0 1.0 2.0 2.0 3.0 3.0 4.0 4.0 5.0
5.0 6.0
POINT_DATA 8
SCALARS sample_scalars float 1
LOOKUP_TABLE my_table
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
LOOKUP_TABLE my_table 8
0.0 0.0 0.0 1.0
1.0 0.0 0.0 1.0
0.0 1.0 0.0 1.0
1.0 1.0 0.0 1.0
0.0 0.0 1.0 1.0
1.0 0.0 1.0 1.0
0.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0

```

Este exemplo representa um cubo com seis faces poligonais, o qual pode ser observado na Figura 2.7. São definidos componentes escalares, normais, e FIELD para as seis faces. Existem também dados escalares associados aos 8 vértices. Um LOOKUP_TABLE de 8 cores, associado aos pontos escalares, também é definido.

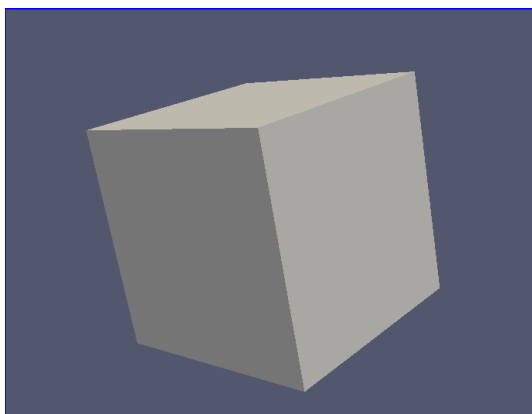


Figura 2.7: Cubo gerado através do arquivo VTK.

Dessa forma, o estudo dos elementos da biblioteca VTK necessários para a implementação do cenário desejado por este projeto é encerrado. A próxima Seção objetiva explicitar os detalhes da implementação dos módulos deste projeto, basicamente os *Web services*, os quais utilizam a biblioteca VTK, e o cliente para consumo dos mesmos, assim como a explanação da arquitetura abordada e como se comunicam os elementos desenvolvidos.

Capítulo 3

Implementação

Após o estudo sobre *Web Services* e da biblioteca VTK, inciou-se a fase de implementação. Foram tomadas como ferramentas de desenvolvimento para *Web Services* a IDE Netbeans, o servidor de aplicações Apache Tomcat 5.5 e o toolkit de desenvolvimento J2EE 5. A implementação do cliente para os *Web Services* propostos foi feita através da IDE NetBeans, o toolkit de desenvolvimento JDK e o ambiente de execução JRE. Para o desenvolvimento da parte computacional e gráfica que os *Web Services* implementam, foi utilizada a biblioteca de classe VTK, compactada em formato .jar, e as dll's específicas necessárias para seu perfeito funcionamento na máquina servidora. Para o servidor de registros UDDI, foi necessário o uso do banco de dados MySQL, o servidor de aplicações Apache Tomcat 5.5 e o framework jUDDI.

Foram construídos neste projeto 4 (quatro) *Web Services*, implantados no servidor Apache Tomcat 5.5, 1 (um) cliente *Desktop* para consumir os *Web Services* e utilizado 1 (um) servidor UDDI implantado no mesmo servidor Apache Tomcat 5.5 com base dados em MySQL.

Os *Web Services* e serviços oferecidos pelos mesmos são:

- PolyDataCreator: Constrói um arquivo de visualização VRML a partir de um arquivo VTK contendo dados do tipo POLYDATA, cor de fundo, opacidade e tipo de interpolação enviados pelo cliente;
- StructuredPointsCreator: Constrói um arquivo de visualização VRML a partir de um arquivo VTK contendo dados do tipo STRUCTURED POINTS, cor de fundo, opacidade e tipo de interpolação enviados pelo cliente;
- UnstructuredGridCreator: Constrói um arquivo de visualização VRML a partir de um arquivo VTK contendo dados do tipo UNSTRUCTURED GRID, cor de fundo, opacidade e tipo de interpolação enviados pelo cliente;
- VRMLConverter: Converte um arquivo VRML para X3D a partir de um arquivo VRML enviado pelo cliente.

Nas Seções que seguem, serão explanados e mostrados os detalhes e particularidades de cada módulo implementado, iniciando com a implementação e caracterização dos *Web services* na linguagem JAVA.

3.1 Web Services em JAVA/NetBeans

3.1.1 Caracterização dos Web Services

Os *Web Services* criados são basicamente projetos Web comuns, no entanto têm a peculiaridade de oferecer e disponibilizar os serviços específicos a qualquer cliente que os deseje utilizar. Particularmente na linguagem JAVA, existem algumas notações específicas para identificar os serviços dentro de um projeto/classe.

- A notação **@WebService** é utilizada para identificar uma classe que contém serviços;
- A notação **@WebMethod** é utilizada para identificar um método disponível como um serviço;
- A notação **@WebParam** identifica os parâmetros dos métodos/serviços.

Para utilizar as notações acima, deve-se importar os seguintes pacotes de classes: `javax.jws.WebService`, `javax.jws.WebMethod` e `javax.jws.WebParam`.

Abaixo segue um trecho da classe `PolyDataService`, do *Web Service* `PolyDataCreator`, fazendo uso das notações citadas.

```
// Identifica uma classe de serviços
@WebService()
public class PolyDataService {

    private byte[] FILE;
    private double RED;
    private double GREEN;
    private double BLUE;
    private double OPACITY;
    private int INTERPOLATION;
    private PD2VRMLCreator vrmlCreator;

    /**
     * Seta o arquivo de entrada .vtk serializado
     */
    // Identifica um serviço oferecido (setFile)
    @WebMethod(operationName = "setFile")
    @Oneway
    // Parâmetro do serviço(_FILE)
    public void setFile(@WebParam(name = "_FILE")
        byte[] _FILE) {
        FILE = _FILE;
    }
}
```

```

/**
 * Seta a cor de fundo da cena
 */
// Identifica um serviço oferecido (setBackgroundColor)
@WebMethod(operationName = "setBackgroundColor")
@Oneway
// Parâmetro do serviço (_RED)
public void setBackgroundColor(@WebParam(name = "_RED")
// Parâmetro do serviço (_GREEN)
double _RED, @WebParam(name = "_GREEN")
// Parâmetro do serviço (_BLUE)
double _GREEN, @WebParam(name = "_BLUE")
double _BLUE) {
    RED = _RED;
    GREEN = _GREEN;
    BLUE = _BLUE;
}

/**
 * Envia o arquivo VRML serializado
 */
// Identifica um serviço oferecido (getVRML)
@WebMethod(operationName = "getVRML")
public byte[] getVRML() {
    try{
        return vrmlCreator.getVRML();
    } catch (Exception ex) {
        Logger.getLogger(PolyDataService.class.getName()).
log(Level.SEVERE, null, ex);
    }
    return null;
}
}

```

Desse modo, ao utilizar a classe descrita anteriormente, a mesma é responsável pela comunicação entre o *Web Service* e o(s) cliente(s), que invoca(m) os métodos explicitados, os quais realizam o envio e recebimento dos parâmetros através do protocolo HTTP, afim de consumir o serviço oferecido pelo *Web Service*.

Uma observação interessante a ser feita é que a classe acima é uma classe JAVA comum que faz uso do J2EE 5. A partir da versão 6 do J2EE, é possível utilizar EJB's (Enterprise Java Beans) como *Web Services*, no entanto, pelo fato do servidor Apache Tomcat 5.5 não suportar o uso do J2EE 6, não foi possível utilizar esse tipo de arquitetura neste servidor.

A seguir, será abordada a comunicação entre os *Web services* e o consumidor de seus serviços, quais os elementos que figuram nesse processo de envio e recebimento de dados, e como são encapsulados estes dados.

3.1.2 Troca de Mensagens

Construídas as classes responsáveis pela troca de informações com os clientes, os *Web Services* implementam o protocolo SOAP, apresentado na Seção 2.1.2. Em síntese, a implementação do protocolo nos leva a um arquivo XML enviado ou recebido tanto pelo cliente quanto pelo *Web Service*, transmitido através do protocolo HTTP ou FTP, a fim de se comunicarem.

Para a classe citada na Seção anterior, um exemplo de mensagem enviada ao *Web Service* pode ser:

```
<?xml version="1.0"encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/»
  <soap:Body>
    <ns0:setBackgroundColor xmlns:ns0="http://service.
polydata.tcc.unesp.br/»
      <_RED>255.0</_RED>
      <_GREEN>0.0</_GREEN>
      <_BLUE>0.0</_BLUE>
    </ns0:setBackgroundColor>
  </soap:Body>
</soap:Envelope>
```

Tal mensagem é uma invocação ao método *setBackground* e os valores 255.0, 0.0 e 0.0 correspondem aos parâmetros *_RED*, *_GREEN* e *_BLUE*, respectivamente.

Para um método com retorno, como no caso do método *getVRML*, também da classe da Seção anterior, segue abaixo um exemplo de uma mensagem SOAP:

```
<?xml version="1.0"encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/»
  <soap:Body>
    <ns0:getVRMLResponse xmlns:ns0="http://service.polydata.
tcc.unesp.br/»
      <return>73616D706C652074657874</return>
    </ns0:getVRMLResponse>
  </soap:Body>
</soap:Envelope>
```

O valor "73616D706C652074657874", que está dentro da *tag return* representa um array de bytes qualquer, o qual poderia ser um arquivo de imagem, ou um arquivo do tipo VTK.

Dessa forma, através de arquivos XML estruturados como os exemplos acima, a troca de mensagens entre cliente e *Web Service* é estabelecida.

Neste projeto, a disposição das trocas de mensagens entre cliente e *Web Services* estão organizadas como segue na Figura 3.1:

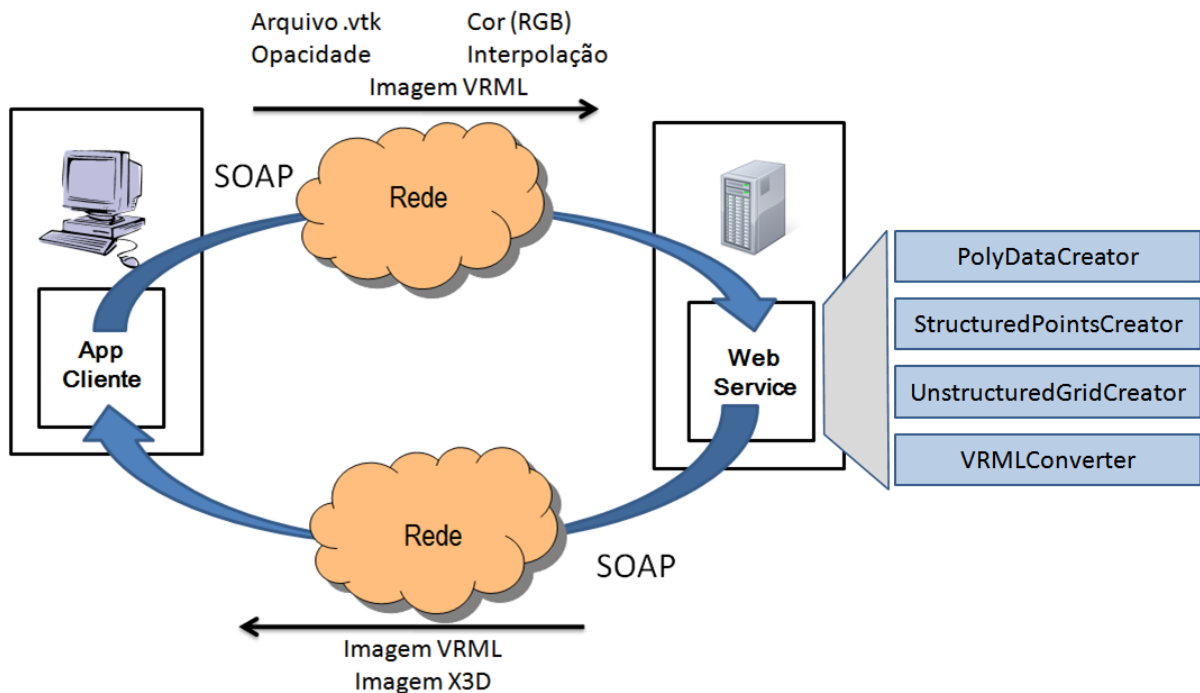


Figura 3.1: Troca de Mensagens.

• **Cliente:**

- envia o arquivo .vtk em forma de array de bytes contendo as informações da imagem a ser gerada;
- envia a cor de fundo (valores do tipo double de R, G e B) da imagem a ser gerada;
- envia o valor em double de opacidade da imagem a ser gerada;
- envia o tipo de interpolação (valor do tipo int) a ser utilizada na imagem a ser gerada;
- recebe a imagem gerada em forma de array de bytes;
- envia uma imagem do tipo VRML na forma de array de bytes e recebe uma imagem do tipo X3D em forma de array de bytes.

• **Web service:**

- recebe o arquivo .vtk em forma de array de bytes contendo as informações da imagem a ser gerada;
- recebe a cor de fundo (valores do tipo double de R, G e B) da imagem a ser gerada;
- recebe o valor em double de opacidade da imagem a ser gerada;

- recebe o tipo de interpolação (valor do tipo int) a ser utilizada na imagem a ser gerada;
- envia a imagem gerada em forma de array de bytes;
- recebe uma imagem do tipo VRML na forma de array de bytes e envia uma imagem do tipo X3D em forma de array de bytes.

Analisado o escopo geral da comunicação entre cliente e serviço, a próxima Seção irá discorrer como e onde solicitar aos *Web Services* o conhecimento de seus serviços, a fim de utilizá-los da maneira correta.

3.1.3 Consumo do serviço e WSDL

Como comentado anteriormente, para o consumo de *Web Services* são necessários uma série de fatores, um deles é saber onde se encontra o serviço que se deseja consumir, e quais os parâmetros e métodos de serviço são oferecidos por ele. O elemento responsável por propiciar tal cenário é o WSDL, discutido na Seção 2.1.1.

Ao se criar um *Web Service*, é gerado, automaticamente, seu WSDL, pois não existe *Web Service* com a ausência do seu WSDL. Para os *Web Services* implementados neste projeto, não é diferente, e os mesmos podem ser acessados através dos endereços:

- <http://localhost:8080/PolyDataCreator/PolyDataService?wsdl>;
- <http://localhost:8080/StructuredPointsCreator/StructuredPointsService?wsdl>;
- <http://localhost:8080/UnstructuredGridCreator/UnstructuredGridService?wsdl>;
- <http://localhost:8080/VRMLConverter/Vrml2x3dService?wsdl>.

Através dos endereços acima, é possível visualizar o documento XML que descreve o WSDL de cada *Web Service*.

Acessando o WSDL de cada *Web Service* citado acima, podemos conhecer quais os métodos de serviço, como e por onde acessá-los e quais tipos de dados são esperados e retornados dos mesmos.

3.1.4 Utilização das classes VTK

Após adicionar ao projeto o pacote jar(*JAVA Archive*) contendo as classes da biblioteca VTK, além de incluídas as *dll's* nas variáveis de ambiente, é iniciado o desenvolvimento das classes responsáveis pelo *pipeline* de renderização.

Pelo fato de a biblioteca VTK ter sido originalmente desenvolvida na linguagem C++, a mesma utiliza métodos nativos da linguagem de origem, portanto deve-se carregar na JVM algumas *dll's* para o perfeito funcionamento das classes da VTK. Tais *dll's* são:

- `vtkCommonJava.dll`;
- `vtkFilteringJava.dll`;

- vtkGenericFilteringJava.dll;
- vtkIOJava.dll;
- vtkImagingJava.dll;
- vtkGraphicsJava.dll;
- vtkVolumeRenderingJava.dll.

Abaixo segue como fica a codificação para a carga das *dll*'s citadas. Tal trecho de código deve aparecer dentro da classe onde se deseja utilizar as classes da biblioteca VTK.

```
static {
    System.loadLibrary("vtkCommonJava");
    System.loadLibrary("vtkFilteringJava");
    System.loadLibrary("vtkGenericFilteringJava");
    System.loadLibrary("vtkIOJava");
    System.loadLibrary("vtkImagingJava");
    System.loadLibrary("vtkGraphicsJava");
    System.loadLibrary("vtkVolumeRenderingJava");
}
```

Concretizada essa parte, é feito o uso das classes e métodos necessários para realizar o *pipeline* de renderização. Diversas pesquisas, estudos e testes foram feitos até se conseguir uma perfeita combinação dos elementos utilizados desde a leitura do arquivo VTK até a exportação da cena proposta em formato VRML.

Para o *Web Service PolyDataCreator*, por exemplo, a classe PD2VRMLCreator utiliza objetos das seguintes classes da VTK:

- **vtkPolyDataReader**: Lê arquivos VTK de estrutura poligonal (Polydata);
- **vtkLookupTable**: Carrega a tabela de cores a ser utilizada na estrutura;
- **vtkActor**: Ator a ser adicionado na cena;
- **vtkProperty**: Propriedades do ator, como opacidade, por exemplo;
- **vtkRenderer**: Renderizador da cena;
- **vtkRenderWindow**: Janela onde será exposta a cena renderizada;
- **vtkRenderWindowInteractor**: Interador da cena com o usuário;
- **vtkVRMLExporter**: Escreve a cena em formato VRML.

O trecho de código a seguir mostra como é utilizado cada elemento citado no método geraVRML() da classe PD2VRMLCreator.

```

// Cria um reader para Poly Data
vtkPolyDataReader reader1 = new vtkPolyDataReader();
// Seleciona o arquivo a ser lido
// O arquivo temp.vtk é um arquivo temporário
// gravado no servidor
reader1.SetFileName("temp.vtk");

// Cria um Lookup Table - Tabela de cores
vtkLookupTable lut = new vtkLookupTable();
// Seta o número de cores possíveis da tabela
lut.SetNumberOfColors(256);
// Seta o intervalo de cores existentes da tabela
lut.SetTableRange(0, 255);
lut.Build(); // Constrói a tabela de cores

// Cria um Mapper para o tipo Poly Data
vtkPolyDataMapper mapper = new vtkPolyDataMapper();
// Pega os pontos do arquivo
mapper.SetInput(reader1.GetOutput());
mapper.SetLookupTable(lut);
mapper.SetColorModeToMapScalars();

// Cria ator para cena
vtkActor actor = new vtkActor();
actor.SetMapper(mapper);

// Seta opacidade da imagem
vtkProperty property = new vtkProperty();
property.SetOpacity(opacity);

// Seta tipo de interpolação da imagem
// 0 - Flat, 1 - Gouraud, 2 - Phong
switch(interpolacao) {
    case 0: property.SetInterpolationToFlat();
            break;
    case 1: property.SetInterpolationToGouraud();
            break;
    case 2: property.SetInterpolationToPhong();
            break;
}

// Adiciona as propriedades de opacidade
// e interpolação ao ator
actor.SetProperty(property);

```



```

// Cria um renderizador
vtkRenderer renderer = new vtkRenderer();
// Seta a cor de fundo da cena no renderizador
renderer.SetBackground(colorR, colorG, colorB);
// Cria uma janela para o renderizador
vtkRenderWindow renWin = new vtkRenderWindow();
renWin.AddRenderer(renderer);
renWin.SetSize(512, 512);

// Cria interação entre o usuário e a cena
vtkRenderWindowInteractor renInteractor =
new vtkRenderWindowInteractor();
renInteractor.SetRenderWindow(renWin);

// Adiciona o ator à cena
renderer.AddActor(actor);
renWin.Render();
renInteractor.SetRenderWindow(renWin);
renInteractor.SetDesiredUpdateRate(1);
renInteractor.Initialize();

renderer.Render();

// Cria o objeto responsável pela exportação da cena para VRML
vtkVRMLExporter vrmlExporter = new vtkVRMLExporter();
// Seta o nome do arquivo a ser gerado
vrmlExporter.SetFileName("temp_vrml.wrl");
// Seleciona a cena a ser exportada
vrmlExporter.SetRenderWindow(renWin);
vrmlExporter.Write(); // Escreve o arquivo VRML

renWin.Finalize();
renderer.Clear();

```

A Figura 3.2 ilustra sucintamente o papel desempenhado pela classe mostrada acima.

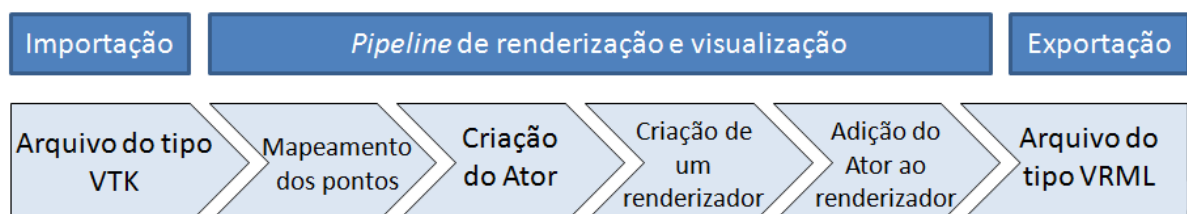


Figura 3.2: Fluxo do processo de geração de um arquivo do tipo VRML a partir de um arquivo do tipo VTK.

A classe mostrada anteriormente, como já dito, refere-se ao *Web Service PolyDataCreator*. Os *Web Services StructuredPointsCreator* e *Unstructured-*

GridCreator têm estrutura parecida, no entanto utilizam alguns objetos diferentes no processo de *pipeline* de renderização, o objeto responsável por fazer a leitura do arquivo VTK, por exemplo, é diferente nos três projetos, como pode ser observado a seguir:

- **PolyDataCreator**

```
// Cria um reader para Poly Data
vtkPolyDataReader reader1 = new vtkPolyDataReader();
reader1.SetFileName("temp.vtk");
```

- **StructuredPointsCreator**

```
// Cria um reader para Structured Points
vtkStructuredPointsReader reader1 =
    newvtkStructuredPointsReader();
reader1.SetFileName("temp.vtk");
```

- **UnstructuredGridCreator**

```
// Cria um reader para Unstructured Grid
vtkUnstructuredGridReader reader1 =
    new vtkUnstructuredGridReader();
reader1.SetFileName("temp.vtk");
```

Outra peculiaridade apresentada, específica para cada tipo de estrutura utilizada para se formar a cena em questão, é o objeto utilizado para mapear os pontos em estruturas do tipo PolyData, diferente para os tipos Structured-Points e UnstructuredGrid, os quais utilizam o mesmo objeto para o mapeamento de pontos.

- **PolyDataCreator**

```
// Cria um Mapper para o tipo Poly Data
vtkPolyDataMapper mapper = new vtkPolyDataMapper();
mapper.SetInput(reader1.GetOutput());
```

- **StructuredPointsCreator e UnstructuredGridCreator**

```
// Cria um Mapper para o tipo Structured Points
vtkDataSetMapper mapper = new vtkDataSetMapper();
mapper.SetInput(reader1.GetOutput());
```

A não ser as especificidades mostradas acima, os outros objetos responsáveis por realizar o *pipeline* de renderização são os mesmos para os três *Web services* implementados que utilizam a biblioteca VTK.

3.1.5 Conversão de arquivos do tipo VRML para arquivos do tipo X3D

Para a implementação do *Web Service* responsável por converter arquivos do tipo VRML para X3D¹, o qual é considerado o padrão sucessor do VRML, foram utilizadas classes prontas, encontradas através de pesquisas realizadas na internet², e disponíveis em um repositório de códigos e projetos de acesso público, o *SourceForge*³.

Com as classes necessárias para a conversão dos arquivos, foi necessário abstrair o essencial das classes a fim de obter somente o suficiente para realizar a conversão de VRML para X3D. Após estudos feitos em cima do código obtido, foi criada uma biblioteca de classes JAVA, possibilitando assim a inclusão das classes no *Web Service* de maneira simplificada.

Este *Web service* não faz uso de nenhum objeto ou funcionalidade disponível pela biblioteca VTK, tem apenas como objetivo converter os arquivos do tipo VRML para X3D. A razão pela qual se faz uso desse processo de conversão está no fato do padrão X3D ser mais novo que o padrão VRML. A biblioteca VTK tem suporte apenas ao padrão VRML (importação/exportação de arquivos), portanto este serviço de conversão, de fato deve ser atribuído a uma entidade externa, fora do escopo da biblioteca, tornando assim esta a justificativa para a implementação deste *Web Service* de conversão.

A Figura 3.3 ilustra a proposta do *Web Service*.

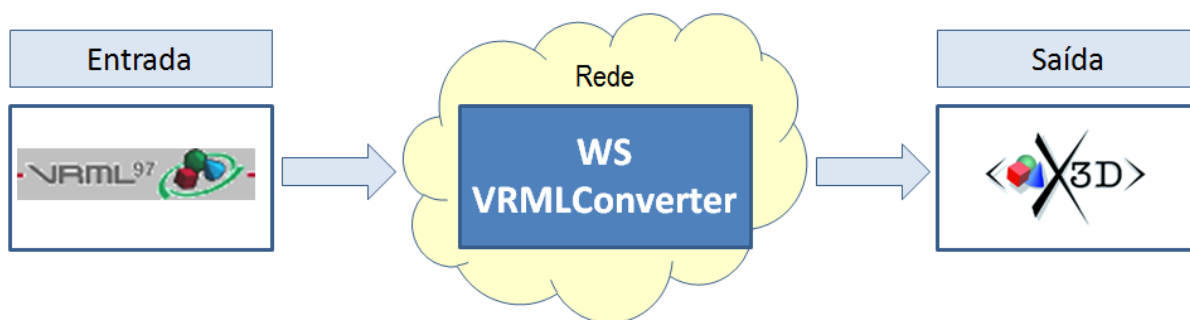


Figura 3.3: Conversão do padrão VRML para X3D.

Este *Web Service*, como já dito, oferece somente o serviço de conversão de arquivos VRML para X3D, para tanto ele possui apenas 2 (dois) métodos de serviço, o método **setFile**, o qual aceita como parâmetro um *array* de bytes, ou seja, o arquivo VRML serializado, e o método **getX3D**, que retorna um *array* de bytes representando o arquivo X3D de forma serializada, ficando assim a cargo do cliente realizar a serialização/desserialização dos arquivos de entrada/saída do *Web Service* em questão.

¹<http://www.web3d.org/x3d>

²Projeto encontrado em http://ovrt.nist.gov/v2_x3d.html

³Disponível em <http://www.sourceforge.net>

Por não se tratar do foco principal deste projeto, detalhes de implementação e codificação das classes deste *Web service* especificamente, não serão abordados com mais detalhes.

3.2 Repositório UDDI

Após desenvolvidos os *Web Services* e implantados no servidor, o consumo dos mesmos por um ou mais cliente está pronto para ser realizado, no entanto, através de estudos feitos na Seção 2.2.4, verificou-se que o uso de um repositório UDDI para o armazenamento e busca dos *Web Services*, afim de conhecer seus respectivos WSDLs se faz necessário. Em razão disso, foi implantado o projeto Apache jUDDI, um repositório UDDI, para efeito de testes e simulações de maneira mais próximas do universo real, em se tratando de *Web Services* e consumo e busca dos mesmos.

3.2.1 Configuração e implantação do repositório Apache jUDDI

Para o uso do repositório jUDDI, alguns requisitos devem ser respeitados. É necessário o uso de um servidor de banco de dados (no caso deste projeto o banco de dados MySQL) e outro servidor de aplicações (neste projeto foi utilizado o Apache Tomcat 5.5). O projeto jUDDI, bem como toda sua documentação e instruções de uso e configuração, pode ser acessado e obtido através do endereço <http://juddi.apache.org>.

Após configurado todo o ambiente de execução do Apache jUDDI, o mesmo pode ser acessado através do endereço <http://localhost:8080/juddi>. As funcionalidades oferecidas pelo jUDDI podem ser acessadas pelo console do mesmo, o qual exibe uma vasta lista de itens relacionados à publicação, remoção e busca de *Web Services*, a seguir são citados os itens disponibilizados pelo console jUDDI:

- **jUDDI API (proprietary)**

- `get_registryInfo`;
- `find_publisher`;
- `get_publisherDetail`;
- `save_publisher` `delete_publisher`.

- **UDDI Inquiry API**

- `find_business`;
- `find_service`;
- `find_binding`;
- `find_tModel`;
- `find_relatedBusinesses`;
- `get_businessDetail`;
- `get_businessDetailExt`;

- get_serviceDetail;
- get_bindingDetail;
- get_tModelDetail.

- **UDDI Publish API**

- get_authToken;
- get_registeredInfo;
- discard_authToken;
- save_business;
- save_service;
- save_binding;
- save_tModel;
- delete_business;
- delete_service;
- delete_binding;
- delete_tModel;
- add_publisherAssertions;
- set_publisherAssertions;
- get_publisherAssertions;
- delete_publisherAssertions;
- get_assertionStatusReport.

Assim como a conversão de arquivos do padrão VRML para o padrão X3D, o foco deste projeto não consiste no estudo do funcionamento e comunicação de um repositório UDDI, portanto a abordagem tomada em relação ao repositório UDDI foi bastante prática e sucinta, atentando basicamente às funcionalidades essenciais oferecidas pelo jUDDI, publicando os WSDLs obtidos a partir dos *Web Services* desenvolvidos.

Foi feito o uso de uma aplicação oferecida gratuitamente, implementada na linguagem JAVA, o UDDIBROWSER⁴, o qual possui uma interface gráfica que facilita bastante o uso das funcionalidades oferecidas por um repositório UDDI, no caso o jUDDI.

3.3 Cliente para o consumo dos serviços

Optou-se, neste projeto, fazer uso de uma aplicação *Standalone* para o consumo dos serviços oferecidos pelos *Web Services* implementados. Aplicações *Web* são mais comuns para o consumo dos serviços, no entanto o procedimento é o mesmo para ambos os casos.

Como dito anteriormente, todo o projeto foi implementado na linguagem JAVA, sendo assim, a execução da aplicação cliente independe do sistema

⁴Disponível para *download* em <http://uddibrowser.org/>

operacional da máquina implantada, bastando apenas a premissa de ter a máquina virtual JAVA devidamente instalada (JRE).

O principal propósito do projeto é observar o comportamento em relação à utilização de *Web Services* no transporte dos objetos gerados. Portanto, se fez necessário medir o tempo gasto no envio do arquivo vtk, renderização do objeto nos *Web Services* e retorno do arquivo VRML ao cliente.

Para tanto, estudos foram realizados a fim de compreender como é implementada a comunicação da aplicação com o *Web Service*. Em razão da complexidade e da implementação dos algoritmos envolvendo a biblioteca VTK ter sido realizada nos *Web Services*, a aplicação cliente foi implementada de maneira bastante simples.

3.3.1 Geração das classes de consumo na aplicação cliente

A partir do endereço do WSDL dos *Web Services* criados, a IDE Netbeans gera as classes necessárias para a comunicação e conexão com os serviços. Ao indicar o endereço do WSDL do *Web Service* PolyDataCreator, por exemplo, o pacote **br.unesp.tcc.polydata.service** é gerado, contendo as seguintes classes:

- CreateVRML.java;
- GetVRML.java;
- GetVRMLResponse.java;
- ObjectFactory.java;
- PolyDataService.java;
- PolyDataServiceService.java;
- SetBackgorundColor.java;
- SetFile.java;
- SetOpacity.java;
- package-info.java.

Pelo o que se pode observar, os nomes dos métodos de serviço do *Web Service* PolyDataCreator são utilizados como nome de algumas classes geradas, por exemplo, o método de serviço **createVRML()** e a classe **CreateVRML.java**, o método **byte[] getVRML()** e a classe **GetVRML.java**.

No entanto, as classes que realmente nos interessam, e que realizam de fato a comunicação com o *Web Service*, são as classes **PolyDataService.java** e **PolyDataServiceService.java**, como pode se observar no trecho de código que segue, retirado da classe FrameUI.java, o qual faz parte da aplicação cliente.

```

// objeto responsável pela comunicação com o WS
PolyDataService portPolyData;
portPolyData = new PolyDataServiceService().
    getPolyDataServicePort();
// seta o arquivo selecionado serializado
portPolyData.setFile(getSerializado());
// seta a cor de fundo
portPolyData.setBackgroundColor(
    jPanel2.getBackground().getRed(),
    jPanel2.getBackground().getGreen(),
    jPanel2.getBackground().getBlue());
// seta a interpoção
portPolyData.setInterpolation(interpolation);
// seta a opacidade
portPolyData.setOpacity(opacity);
// invoca o método responsável por gerar a cena
portPolyData.createVRML();
// recebe o arquivo VRML serializado do WS
vrml = portPolyData.getVRML();

```

Dessa forma, podemos observar que o objeto **portPolyData** é responsável por invocar e consumir os métodos de serviços oferecidos pelo *Web Service PolyDataCreator*.

A seguir, as Figuras 3.4 e 3.5 ilustram como estão estruturados os pacotes e classes gerados pelo cliente a partir dos endereços WDSL de cada *Web Service* utilizado.

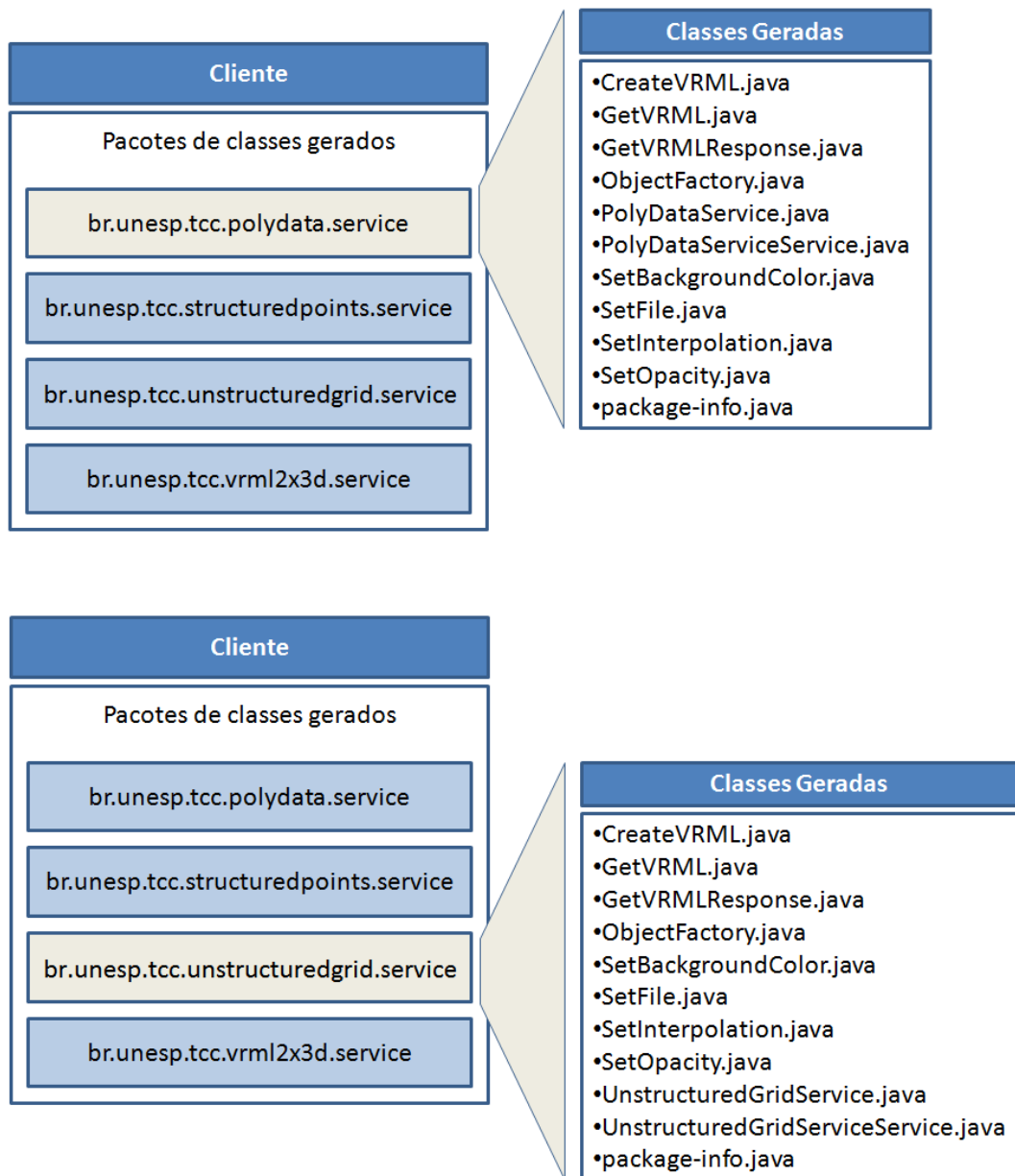


Figura 3.4: Pacotes e classes gerados pelo cliente para consumo dos *Web Services* PolyDataCreator e UnstructuredGridCreator.

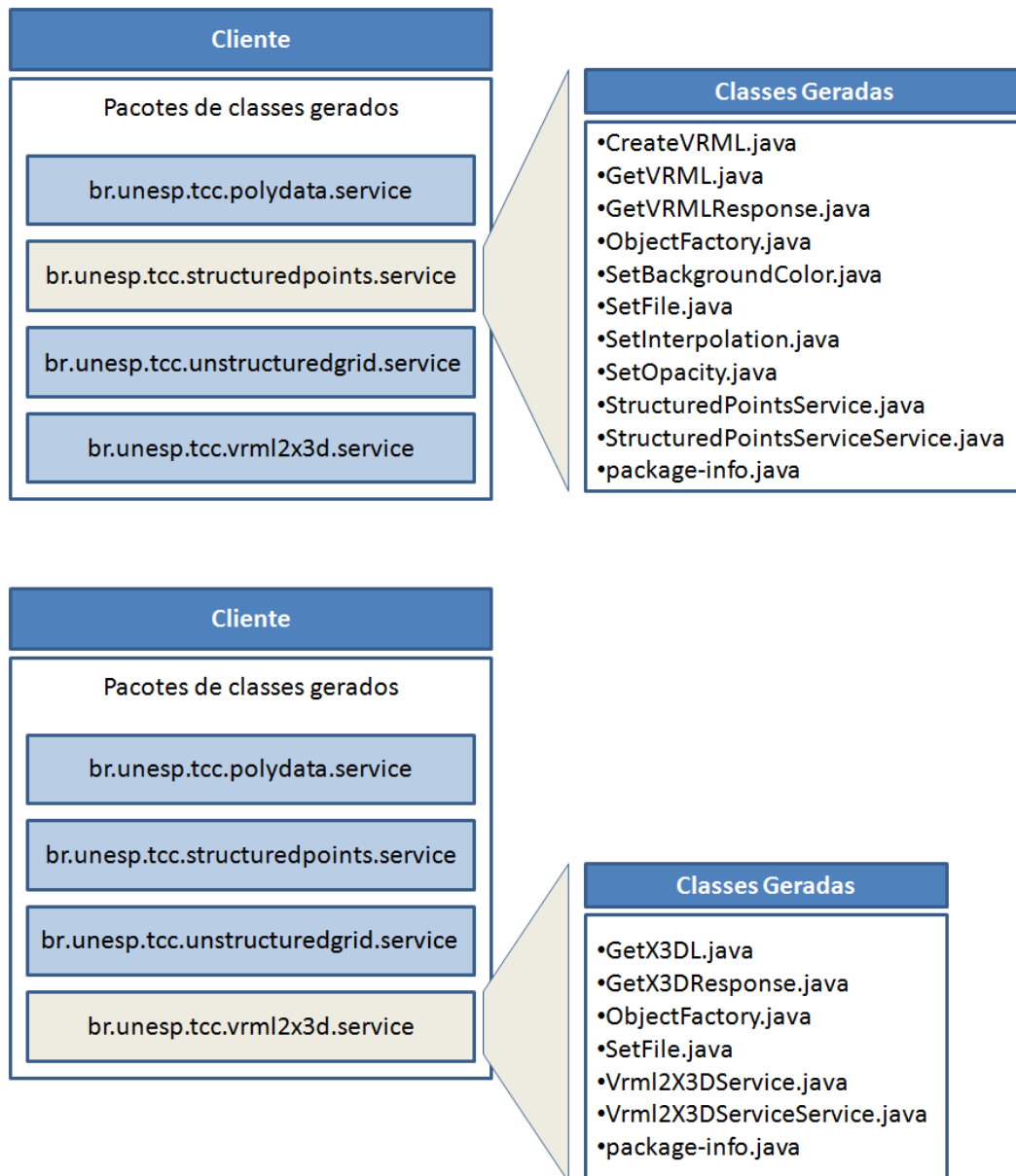


Figura 3.5: Pacotes e classes gerados pelo cliente para consumo dos *Web Services* StructuredPointsCreator e VRMLConverter.

3.3.2 Envio dos arquivos do tipo VTK

Um detalhe interessante a respeito da comunicação entre *Web Service* e cliente, é que, em razão de ambos poderem ser escritos em linguagens diferentes, como já foi comentado anteriormente, a troca de dados entre os mesmos deve ser feita através de tipos comuns, como inteiros, caracteres, bytes, booleanos, entre outros. Devido à esse cenário, o envio e recebimento dos arquivos envolvidos nas trocas de mensagens passaram por uma transformação para *array* de bytes, a esse processo se dá o nome de serialização, no qual o resultado é o que se chama de objeto serializado, ou seriado.

Para tanto, o método **getSerializado** foi criado, com o intuito de serializar o arquivo do tipo *vtk*, selecionado pelo usuário, e então, a partir deste arquivo serializado, enviá-lo ao *Web Service* para que o mesmo possa ser lido e

realizado o processo de renderização da cena.

A seguir, é mostrado o método responsável por serializar os arquivos do tipo vtk.

```
public byte[] getSerializado() throws Exception {
    // inputFile é o arquivo selecionado pelo usuário
    FileInputStream file = new FileInputStream(inputFile);
    byte[] aux = new byte[1024];
    byte[] total = null;
    byte[] buffer = null;
    int cont;
    while ((cont = file.read(aux)) > 0) {
        if (total != null) {
            buffer = new byte[total.length + cont];
            for (int pos = 0; pos < total.length; pos++) {
                buffer[pos] = total[pos];
            }
            for (int pos = 0; pos < cont; pos++) {
                buffer[pos+total.length] = aux[pos];
            }
            total=buffer;
        } else {
            total = aux;
        }
        aux = new byte[1024];
    }
    file.close();
    return total;
}
```

Assim, pode-se observar que o arquivo fica armazenado em um *array* de bytes (variável **total**), o qual é retornado ao objeto que invocou o método.

A utilização do mesmo se dá trecho abaixo:

```
// seta o arquivo selecionado
serializado
portPolyData.setFile(getSerializado());
```

Onde o retorno da invocação do método **getSerializado** (arquivo VTK serializado) é enviado como parâmetro para o *Web Service* através do método **setFile**.

3.3.3 Interface gráfica da aplicação

A interface gráfica da aplicação cliente é mostrada através da Figura 3.6.

É possível observar na interface gráfica que é possível gerar imagens a partir de arquivos .VTK contendo os tipos Poly Data, Structuerd Points e Unstructured Grid, que obviamente, referem-se aos *Web Services* desenvolvidos.

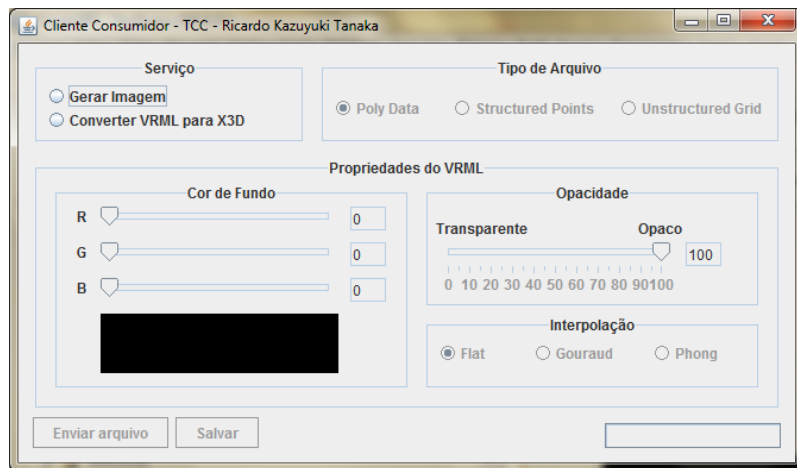


Figura 3.6: Interface gráfica da aplicação cliente.

Além da opção em escolha de qual tipo de estrutura será enviada, a aplicação oferece também opções de escolha da cor de fundo da cena, tipo de interpolação na renderização dos objetos da cena (Flat, Gouraud ou Phong) e nível de transparência/opacidade dos objetos da cena. Outra opção que a aplicação oferece é a conversão de arquivos do tipo VRML para arquivos do tipo X3D, utilizando o *Web Service VRMLConverter*. Após feita a escolha do serviço solicitado e enviado o arquivo referente, é habilitada a opção de salvar o arquivo recebido pelo *Web Service*.

A seguir, no próximo Capítulo, serão mostrados os testes realizados no ambiente que foi desenvolvido.

Capítulo 4

Impacto na comunicação: testes e resultados

Este capítulo é destinado aos testes de impacto da comunicação em rede realizados utilizando os módulos implementados deste projeto. Serão analisados os resultados obtidos a partir de testes realizados em 3 situações diferentes, a primeira, localmente em um único computador(servidor e cliente), a segunda alocando a aplicação cliente em um computador conectado à mesma rede do computador servidor(utilizado na situação anterior) através de um *switch* e sem tráfego algum na rede, e a terceira situação, adicionando outro computador à mesma rede da configuração anterior afim de gerar tráfego na rede, através de *pings* de rede e cópia/exclusão de arquivos no computador servidor. A Figura 4.1 ilustra as situações do ambiente de teste, descritas.

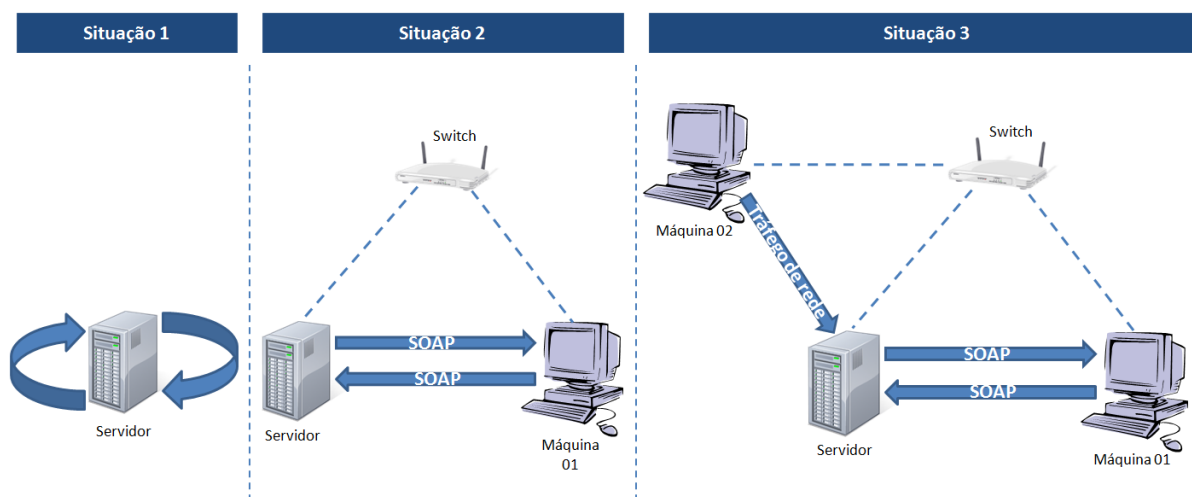


Figura 4.1: Situação criada para realizar os testes.

Para a terceira situação, foi gerado um arquivo *.bat* executado no terceiro computador afim de gerar tráfego na rede. O script criado é apresentado a seguir, na qual a unidade representada por "z:" corresponde à unidade de rede mapeada no computador local, e a unidade representada por "f:" é uma unidade de disco do computador local.

```
:LOOP
IF EXIST z:\trafego.bin (GOTO DELETA)
ELSE (GOTO COPIA)

:DELETA
del z:\trafego.bin

:COPIA
copy f:\trafego.bin z:\trafego.bin

GOTO LOOP
```

Basicamente o script citado verifica se existe o arquivo "trafego.bin", caso exista, apaga o arquivo, caso contrário, copia o arquivo. A execução é realizada em *loop* infinito. Em conjunto com a execução do arquivo .bat gerado através do script, foi executado o comando `ping -t <IP_DO_SERVIDOR>`, o qual envia um *ping* de rede infinitamente ao endereço ip indicado.

A configuração da máquina considerada servidor, em que foram realizados os testes é um notebook equipado com um processador dual-core modelo T4300 2.10GHz e 2GB de memória RAM DDR2, com o sistema operacional Windows 7.

Para a visualização dos arquivos do tipo VRML foi necessária a instalação de um *plugin* de visualização, particularmente neste projeto, foi adotado o *plugin* Cortona3D¹, o qual possui licença de uso gratuito. O mesmo é instalado no browser preferencial do usuário.

A visualização dos arquivos do tipo X3D também requer uma ferramenta adicional de visualização, e neste projeto foi utilizado o Octaga Player², também com licença de uso gratuito, no entanto o mesmo é uma aplicação, e não um *plugin* para browser.

Um resultado interessante que deve ser considerado é que na primeira chamada da aplicação aos *Web services*, o tempo de execução é maior que o tempo médio esperado, a razão disso pode ser levada a estudos futuros considerando o ambiente utilizado nno desenvolvimento e testes. Portanto, as primeiras execuções serão descartadas, a fim de não haver divergências nos resultados dos testes.

As tabelas e gráficos que seguem já estão descartando a primeira execução de cada bateria de testes realizados. Os testes foram realizados em 2 dias diferentes, as execuções dos testes foram realizadas da mesma maneira, porém apenas foram separados em dias diferentes.

O tempo é medido entre o envio do arquivo .vtk ao Web Service e o recebimento da imagem renderizada no cliente. Para todos os testes, o tempo é medido em segundos.

4.1 Testes com PolyData

A primeira bateria de testes foi realizada localmente (situação 1), em seguida na situação 2 e após, na situação 3. Foi utilizado um arquivo do tipo VTK

¹Disponível para download em <http://www.cortona3d.com/>

²Disponível para download em <http://www.octaga.com/>

contendo uma estrutura PolyData, o nome do arquivo é `fran_cut.vtk`, o qual possui 26.460 pontos do tipo `float` e ocupa 2,235 MB de espaço em disco. A cena gerada a partir deste arquivo pode ser observada através da Figura 4.2, que é um arquivo com extensão `.wrl`, do tipo VRML e ocupa 3,947 MB de espaço em disco.

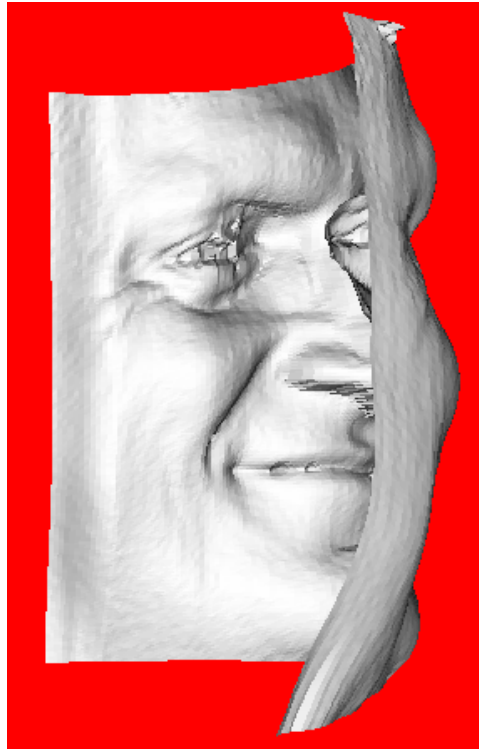


Figura 4.2: Cena renderizada a partir do arquivo `fran_cut.vtk`.

Foram considerados 10 resultados de teste (como dito anteriormente, descartou-se a primeira execução), que podem ser observados na Figura 4.3.

Teste PolyData - <code>fran_cut.vtk</code> - Dia 1			
	Teste Local	Teste Rede Livre	Teste Rede com Tráfego
Execução 01	00:19	00:36	00:51
Execução 02	00:19	00:37	01:23
Execução 03	00:18	00:34	01:28
Execução 04	00:19	00:34	00:54
Execução 05	00:18	00:34	00:45
Execução 06	00:18	00:34	00:48
Execução 07	00:19	00:35	01:09
Execução 08	00:18	00:37	01:00
Execução 09	00:18	00:37	00:53
Execução 10	00:18	00:35	01:03
Média	00:18	00:35	01:01

Figura 4.3: Resultado do primeiro dia de testes com o arquivo `fran_cut.vtk`. Os resultados são medidos em segundos

Os resultados apresentados acima, podem ser visualizados graficamente através da Figura 4.4.

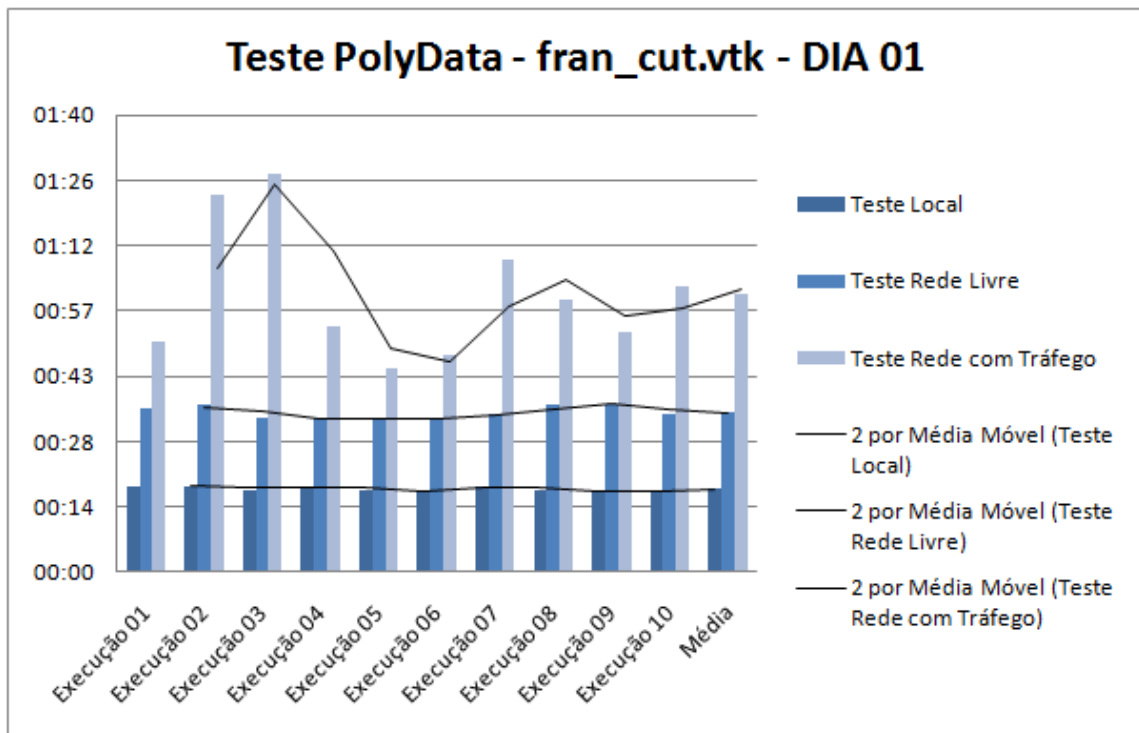


Figura 4.4: Gráficos dos resultados do primeiro dia de testes com o arquivo `fran_cut.vtk`.

Para o mesmo arquivo `fran_cut.vtk` foram realizados testes da mesma natureza e no mesmo ambiente de testes no dia seguinte. Os resultados podem ser observados abaixo, através da Figura 4.5, e da Figura 4.6.

Teste PolyData - fran_cut.vtk - Dia 2			
	Teste Local	Teste Rede Livre	Teste Rede com Tráfego
Execução 01	00:22	00:38	00:41
Execução 02	00:19	00:38	00:43
Execução 03	00:19	00:36	01:05
Execução 04	00:19	00:35	01:08
Execução 05	00:19	00:38	00:52
Execução 06	00:19	00:38	01:19
Execução 07	00:19	00:37	00:41
Execução 08	00:19	00:37	00:46
Execução 09	00:19	00:35	00:53
Execução 10	00:19	00:37	00:41
Média	00:19	00:36	00:52

Figura 4.5: Resultado do segundo dia de testes com o arquivo `fran_cut.vtk`. Os resultados são medidos em segundos

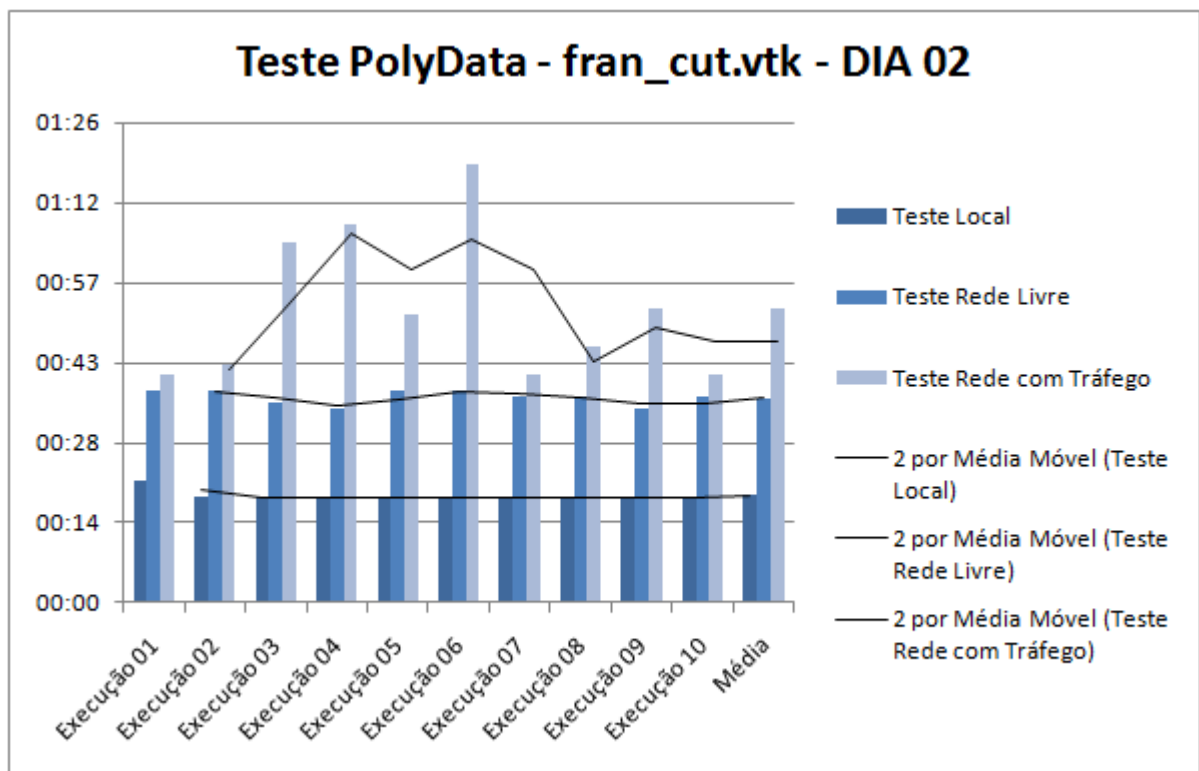


Figura 4.6: Gráficos dos resultados do segundo dia de testes com o arquivo `fran_cut.vtk`.

4.2 Testes com StructuredPoints

A seguir, serão mostrados os testes realizados com o arquivo `ironProt.vtk`, que possui 314.432 pontos do tipo `unsigned char` e ocupa 308KB de espaço em disco. Assim como na seção anterior, os testes foram realizados em 2 dias e em 3 situações diferentes, mantendo as mesmas configurações e condições dos testes anteriores, medindo o tempo total de execução do processo.

O resultado da cena renderizada pode ser visto através da Figura 4.7, que possui extensão de arquivo `.wrl` e ocupa 2,332MB de espaço em disco.

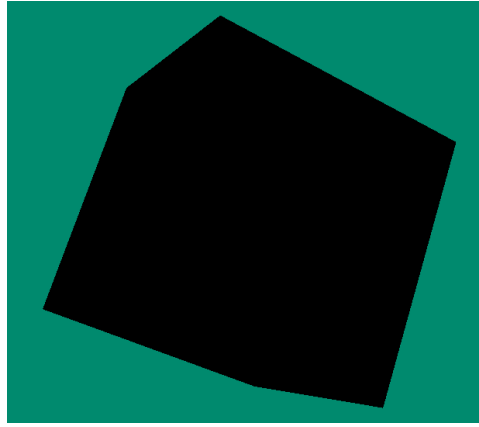


Figura 4.7: Cena renderizada a partir do arquivo ironProt.vtk.

A Figura 4.8 e os gráficos da Figura 4.9 referem-se aos testes realizados no primeiro dia.

Teste StructuredPoints - ironProt.vtk - Dia 1			
	Teste Local	Teste Rede Livre	Teste Rede com Tráfego
Execução 01	00:07	00:11	00:14
Execução 02	00:07	00:10	00:21
Execução 03	00:07	00:11	00:16
Execução 04	00:07	00:09	00:19
Execução 05	00:07	00:09	00:19
Execução 06	00:07	00:12	00:26
Execução 07	00:07	00:11	00:19
Execução 08	00:07	00:10	00:15
Execução 09	00:07	00:11	00:13
Execução 10	00:07	00:11	00:18
Média	00:07	00:10	00:18

Figura 4.8: Resultado do primeiro dia de testes com o arquivo ironProt.vtk. Os resultados são medidos em segundos

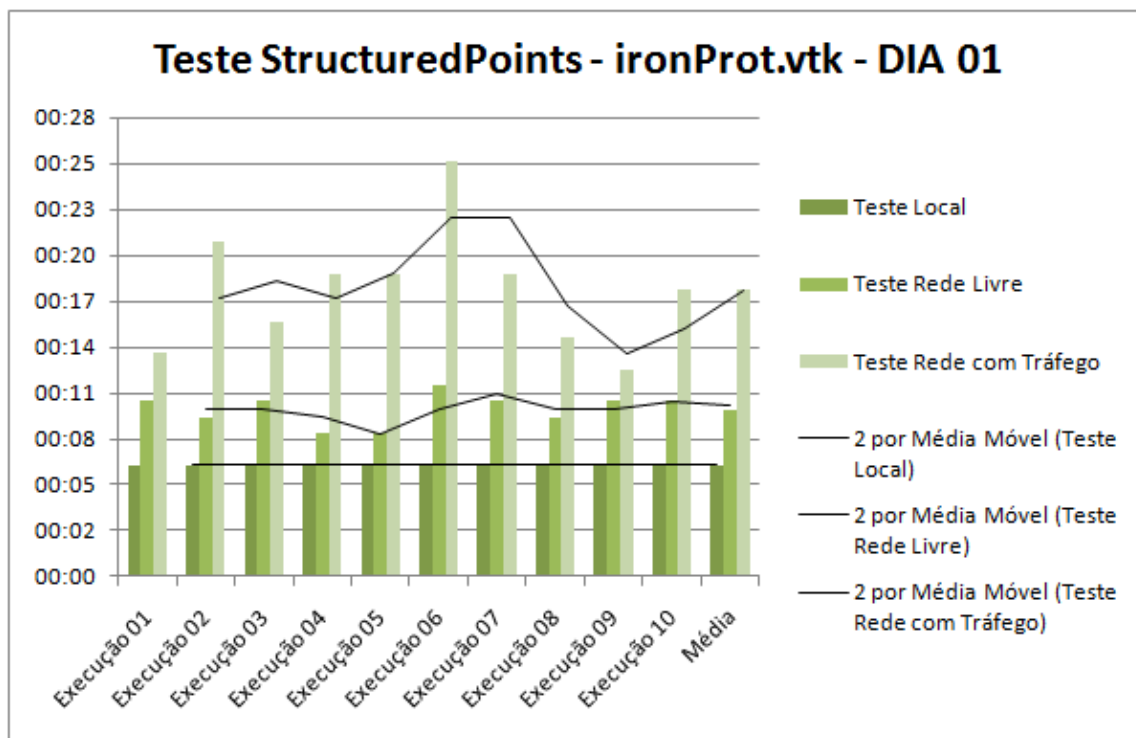


Figura 4.9: Gráficos dos resultados do primeiro dia de testes com o arquivo ironProt.vtk.

As Figuras 4.10 e 4.11 ilustram os resultados obtidos a partir da bateria de testes realizada no segundo dia.

Teste StructuredPoints - ironProt.vtk - Dia 2			
	Teste Local	Teste Rede Livre	Teste Rede com Tráfego
Execução 01	00:08	00:15	00:12
Execução 02	00:07	00:09	00:14
Execução 03	00:06	00:12	00:12
Execução 04	00:06	00:09	00:14
Execução 05	00:06	00:10	00:24
Execução 06	00:06	00:10	00:39
Execução 07	00:07	00:10	00:17
Execução 08	00:06	00:12	00:21
Execução 09	00:06	00:09	00:15
Execução 10	00:07	00:12	00:15
Média	00:06	00:10	00:18

Figura 4.10: Resultado do segundo dia de testes com o arquivo ironProt.vtk. Os resultados são medidos em segundos

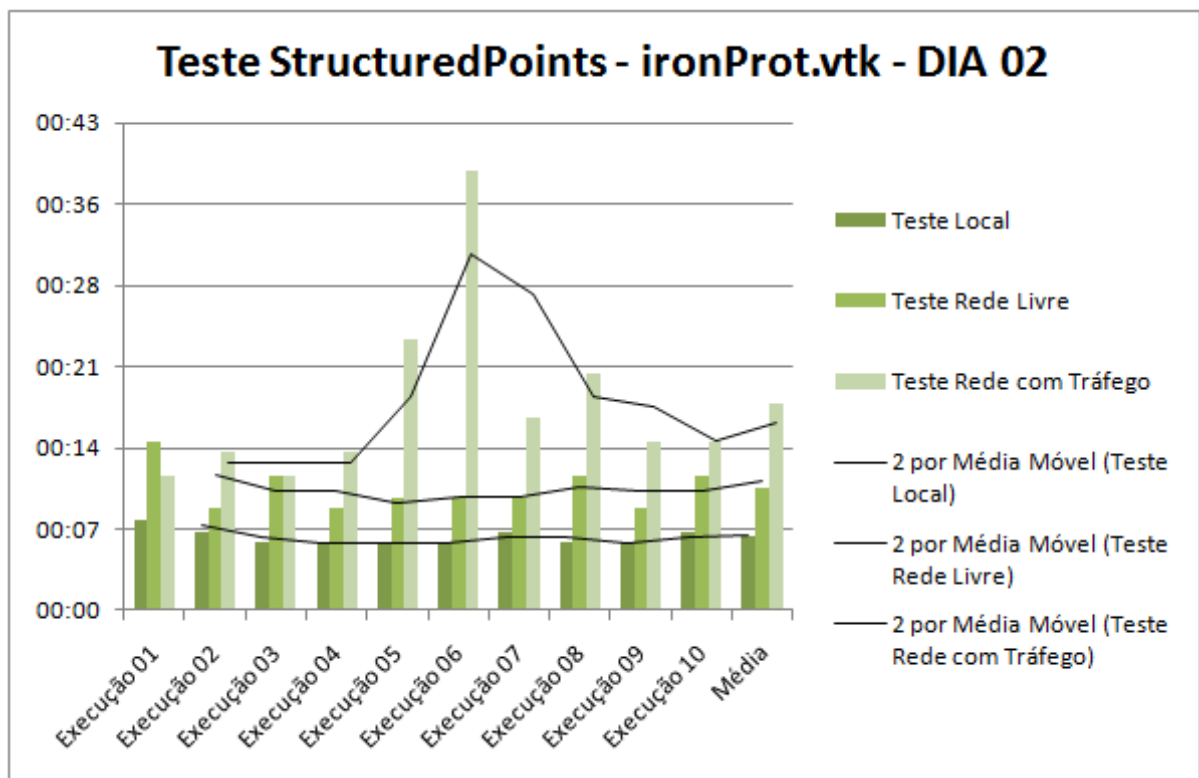


Figura 4.11: Gráficos dos resultados do segundo dia de testes com o arquivo ironProt.vtk.

4.3 Análise dos resultados

Os gráficos das Figuras 4.4, 4.6, 4.9 e 4.11, mostram claramente, como esperado, que o tempo aumenta da situação 1 para situação 2, e desta para a situação 3.

Estes mesmos gráficos permitem tecer outras observações. Considerando os dados referentes às colunas "Teste Local" e "Teste Rede Livre", a variação dos tempos de uma execução para a outra é bastante pequena, isto é, o desvio padrão é baixo, assim como, a diferença entre os valores dos tempos máximo e mínimo é pequena, como mostra a Figura 4.12 e a Figura 4.13. Nos gráficos, isto fica evidente na pequena oscilação da média móvel.

Teste PolyData - fran_cut.vtk - Dia 1				Teste PolyData - fran_cut.vtk - Dia 2		
	Teste Local	Teste Rede Livre	Teste Rede com Tráfego	Teste Local	Teste Rede Livre	Teste Rede com Tráfego
Execução 01	00:19	00:36	00:51	00:22	00:38	00:41
Execução 02	00:19	00:37	01:23	00:19	00:38	00:43
Execução 03	00:18	00:34	01:28	00:19	00:36	01:05
Execução 04	00:19	00:34	00:54	00:19	00:35	01:08
Execução 05	00:18	00:34	00:45	00:19	00:38	00:52
Execução 06	00:18	00:34	00:48	00:19	00:38	01:19
Execução 07	00:19	00:35	01:09	00:19	00:37	00:41
Execução 08	00:18	00:37	01:00	00:19	00:37	00:46
Execução 09	00:18	00:37	00:53	00:19	00:35	00:53
Execução 10	00:18	00:35	01:03	00:19	00:37	00:41
Desvio	00:00	00:01	00:14	00:00	00:01	00:13
Máximo	00:19	00:37	01:28	00:22	00:38	01:19
Mínimo	00:18	00:34	00:45	00:19	00:35	00:41

Figura 4.12: Tabela de análise dos testes com PolyData.

Teste StructuredPoints - ironProt.vtk - Dia 1				Teste StructuredPoints - ironProt.vtk - Dia 2		
	Teste Local	Teste Rede Livre	Teste Rede com Tráfego	Teste Local	Teste Rede Livre	Teste Rede com Tráfego
Execução 01	00:07	00:11	00:14	00:08	00:15	00:12
Execução 02	00:07	00:10	00:21	00:07	00:09	00:14
Execução 03	00:07	00:11	00:16	00:06	00:12	00:12
Execução 04	00:07	00:09	00:19	00:06	00:09	00:14
Execução 05	00:07	00:09	00:19	00:06	00:10	00:24
Execução 06	00:07	00:12	00:26	00:06	00:10	00:39
Execução 07	00:07	00:11	00:19	00:07	00:10	00:17
Execução 08	00:07	00:10	00:15	00:06	00:12	00:21
Execução 09	00:07	00:11	00:13	00:06	00:09	00:15
Execução 10	00:07	00:11	00:18	00:07	00:12	00:15
Desvio	00:00	00:00	00:03	00:00	00:01	00:08
Máximo	00:07	00:12	00:26	00:08	00:15	00:39
Mínimo	00:07	00:09	00:13	00:06	00:09	00:12

Figura 4.13: Tabela de análise dos testes com StructuredPoints.

Já para a situação 3 (coluna "Teste Rede com Tráfego") em ambos os testes, a oscilação da média móvel é grande, com variações abruptas no tempo entre uma execução e outra. O desvio padrão e os valores máximo e mínimo para o tempo (ver Figuras 4.12 e 4.13) reforçam esta conclusão. Embora os testes tenham sido realizados em uma rede local em um cenário controlado, é lógico concluir que a imprevisibilidade no tempo de execução em uma rede de computadores é um ponto importante na construção de aplicações deste tipo.

Um outro ponto a destacar é que as aplicações implementadas (*Web Services* e cliente) não necessitaram de nenhuma adaptação para serem executadas nas 3 situações, como ocorre com outras aplicações que rodam de forma distribuída. As características de baixo acoplamento e alta coesão, possibilitadas por *Web Services*, permitem o aproveitamento de código e a alteração de trechos internos sem efeitos colaterais para os clientes consumidores dos serviços.

Os testes permitiram observar o impacto da comunicação pela rede no uso dos *Web Services* implementados. Portanto, além da vantagem do aproveitamento de código, por exemplo, o ganho no poder de processamento precisa compensar este impacto.

Capítulo 5

Considerações finais

Um ponto bastante favorável no uso da arquitetura de *Web services* é a portabilidade das aplicações envolvidas, a linguagem de programação utilizada, tanto na implementação dos *Web services* quanto na implementação do cliente, foi JAVA, no entanto a arquitetura de *Web services* permite que as aplicações sejam escritas em linguagens de programação diferentes, graças aos componentes e protocolos utilizados e estudados neste projeto, como o SOAP e o WSDL. Esta característica pode ser alvo de trabalhos futuros, propondo a implementação de *Web services* e clientes escritos em diferentes linguagens de programação e executando em sistemas operacionais distintos.

Como evidenciado neste projeto, o uso da arquitetura dos *Web services* se tornaria muito mais útil em ambientes de rede, nos quais se têm servidores com alto poder de processamento. A implementação e execução de *Web services* semelhantes aos implementados neste projeto em um ambiente de rede com alto desempenho e poder de processamento, *clusters*, por exemplo, pode se tornar fruto de futuros trabalhos, tentando dessa forma, minimizar ao máximo o tempo de renderização de uma cena.

Ao longo do projeto, foram encontradas muitas dificuldades, os principais obstáculos enfrentados foram nas configurações das ferramentas utilizadas, algumas incompatibilidades na interação das ferramentas também fizeram com que o projeto não evoluísse em certas ocasiões, como no caso da utilização do servidor de aplicações GlassFish 3 em conjunto do repositório jUDDI. Outra incompatibilidade apresentada foi no modo em que foram implementados os primeiros *Web services*, os quais utilizavam a arquitetura de EJB's, e que em razão da migração do servidor de aplicações GlassFish 3 para o servidor Apache Tomcat 5.5 passaram a ser incompatíveis, e portanto tiveram de ser reescritos de maneira compatível.

As experiências obtidas ao longo do projeto, documentadas nesta monografia, foram bastante válidas, e agregaram valores e conceitos os quais se estenderão em possíveis trabalhos futuros. Todos os resultados e pesquisas relevantes e interessantes ao autor podem ser encontrados nos Capítulos deste documento.

O objetivo de verificar o impacto da comunicação pela rede no uso de *Web Services* foi atingido. Vários módulos, cliente e servidor, foram implementados e um ambiente controlado para testes com diferentes situações foi definido. Os testes mostraram, que *Web Services*, como outras aplicações com computação distribuída, podem ser executados de forma local ou distribuída, e que

o tráfego pela rede é um importante requisito a ser considerado.

Referências Bibliográficas

Kitware public wiki - vtk. 2010.

Disponível em <http://www.vtk.org/Wiki/VTK>

BARRY, D. K. *Web services and service-oriented architecture*. Morgan Kaufmann Publishers, 2003.

CERAMI, E. *Web services essentials*. Manning Publications Co., 2006.

CHAPPELL, D.; JEWELL, T. *Java web services*. O'Reilly & Associates, 2002.

CHRISTENSEN, E.; CURBERA, F.; MEREDITH, G.; S., W. *Web services description language (wsdl) 1.1*. Relatório Técnico, W3C Working Group, Ariba, International Business Machines Corporation, Microsoft, 2001.

ENGLANDER, R. *Java and soap*. O'Reilly & Associates, 2002.

FERRIS, C.; M., C.; HOLLANDER, D. *Web services architecture*. Relatório Técnico, W3C Working Group, Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University, 2004.

JOSUTTIS, N. M. *Soa in practice*. O'Reilly & Associates, 2007.

LAW, C. C. *The vtk user's guide*. Kitware, Inc., 2005.

MUELLER, J. P. *Special edition using soap*. QUE, 2002.

PULIER, E.; TAYLOR, H. *Understanding enterprise soa*. O'Reilly & Associates, 2002.

SCHROEDER, W. J.; MARTIN, K. M. *The visualization handbook*. Elsevier ButterworthHeinemann, 2005.

SHU, G.; CHEN, D. Web service-oriented visualization for engineering data. *International Conference on Advanced Language Processing and Web Information Technology*, 2008.

Apêndice A

Configuração do servidor de aplicações Apache Tomcat 5.5

Antes de iniciar o desenvolvimento dos *Web Services*, deve-se fazer uma preparação prévia do servidor em que os *Web Services* serão alocados. Foi escolhido o servidor de aplicações Apache Tomcat 5.5¹ para a implantação dos *Web Services*, como dito anteriormente. Após feita a instalação padrão e inicialização do servidor, o mesmo poderá ser acessado através de algum browser pelo endereço <http://localhost:8080>. Uma página como a apresentada pela Figura A.1 deve ser carregada:

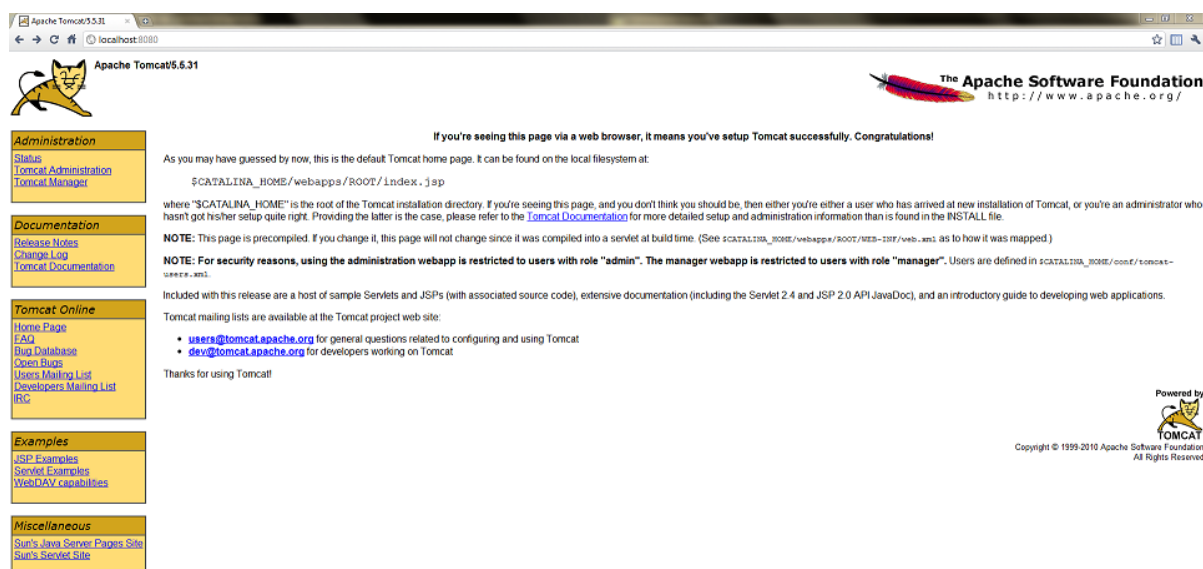


Figura A.1: Página inicial do Apache Tomcat 5.5.

A fim de possibilitar a administração do servidor, a ferramenta de administração deve ser baixada e implantada como seguem as informações disponíveis no endereço <http://tomcat.apache.org>.

Uma tela como a apresentada na figura A.2 deve aparecer, ao acessar o endereço <http://localhost:8080/admin> e efetuar o *login*.

Outra ferramenta interessante disponível no servidor Apache Tomcat 5.5 é o Tomcat manager, o qual possibilita a implantação/desimplantação, inicialização/interrupção de projetos Web. Diferente da ferramenta administrativa

¹Disponível para download em <http://tomcat.apache.org>

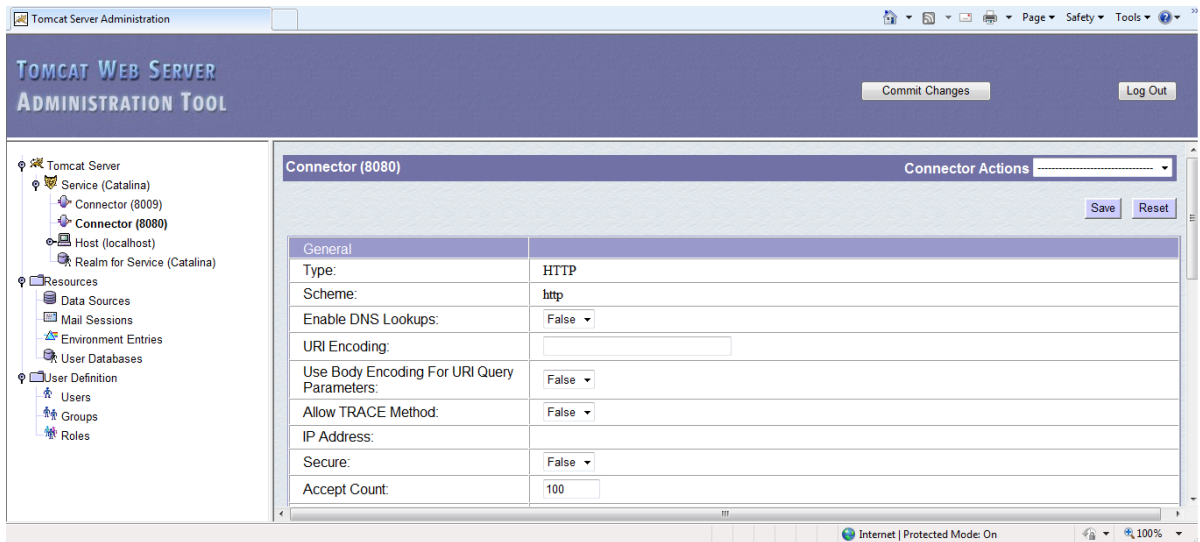


Figura A.2: Página da ferramenta administrativa do Apache Tomcat 5.5.

do servidor, o Tomcat manager não necessita ser baixado e implantado no servidor, pois o mesmo vem já embutido no servidor. Para acessá-lo, basta acessar o endereço <http://localhost:8080/manager>. Feito isso, após efetuado o *login*, uma tela como a figura A.3 deverá ser mostrada

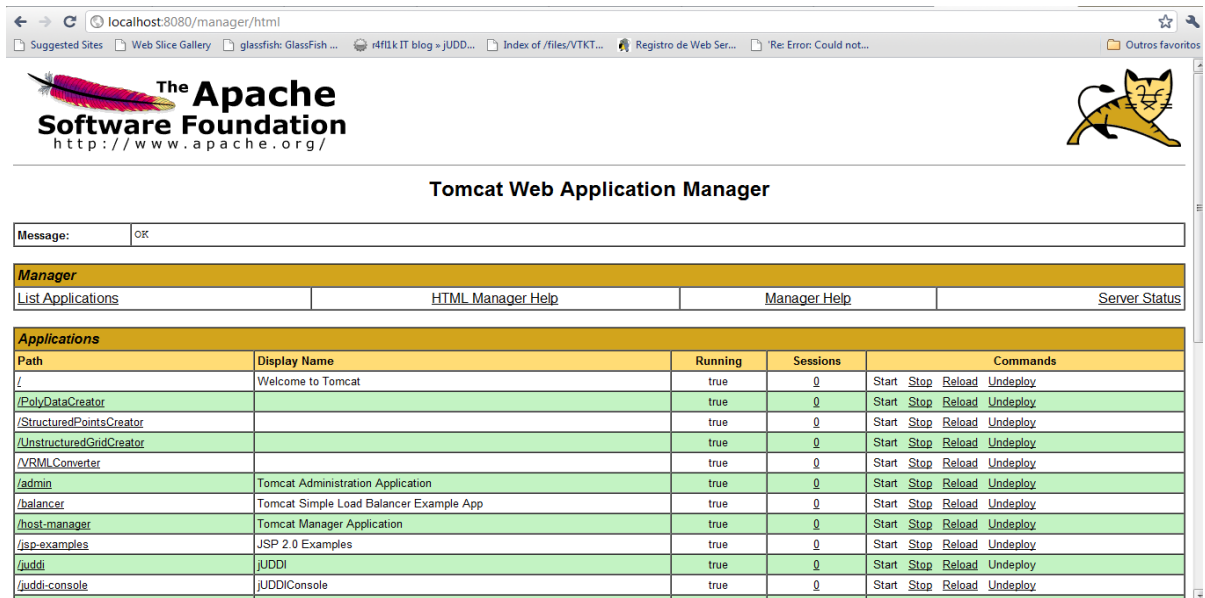


Figura A.3: Página do Tomcat Manager.

Apêndice B

Criação de *Web services* utilizando a IDE NetBeans 6.9.1

Desenvolver um *Web service* utilizando a IDE NetBeans consiste em uma sequência de passos bastante intuitiva e fácil. Primeiramente, é preciso criar um projeto Web comum. Ao escolher a opção de novo projeto, deve-se navegar na janela apresentada conforme a Figura B.1.

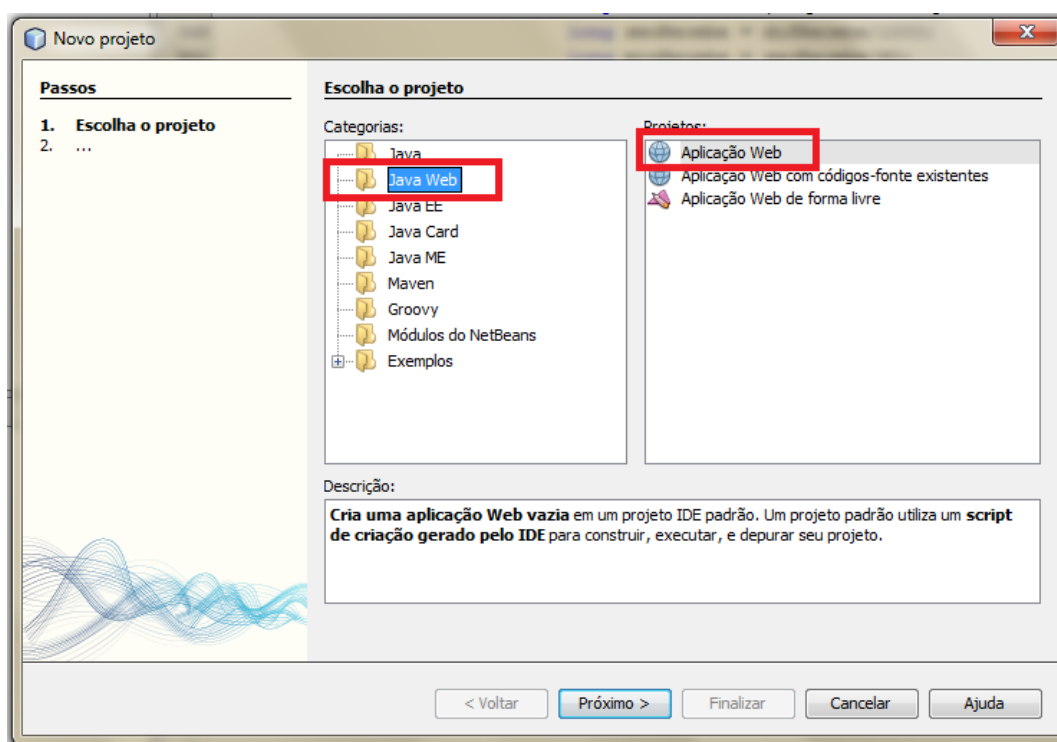


Figura B.1: Criação de um projeto Web comum.

Ao clicar em "Próximo", uma tela, como a apresentada abaixo pela Figura B.2, será mostrada, onde é possível escolher o local onde o projeto será criado e seu nome.

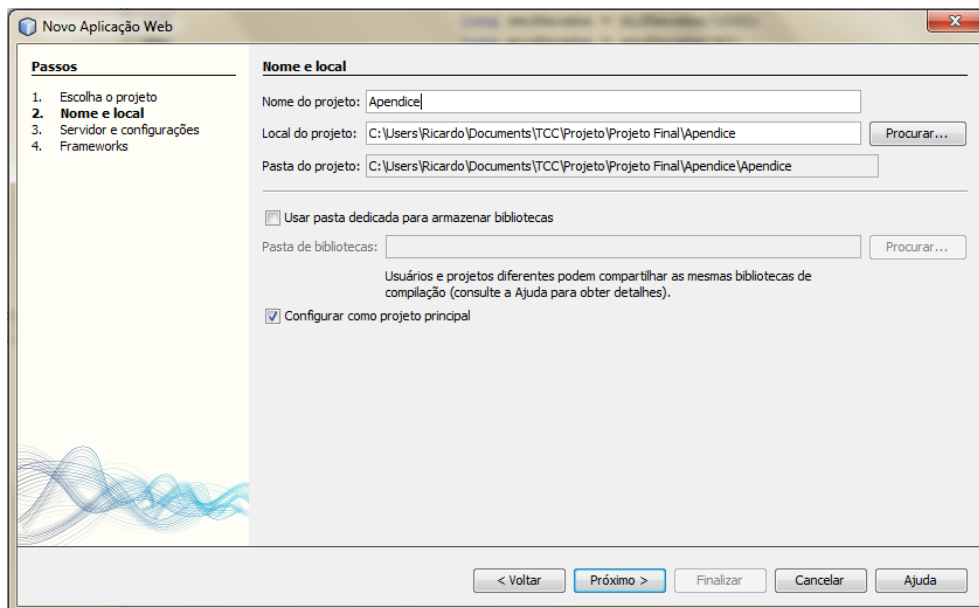


Figura B.2: Local e nome do projeto Web.

Após escolhido o nome e local do projeto e clicado em "Próximo", surgirá a tela representada pela Figura B.3, onde se deve escolher qual servidor de aplicações utilizar (neste caso o Tomcat 5.5) e qual versão do J2EE será utilizada pelos códigos-fonte (1.4 neste caso).

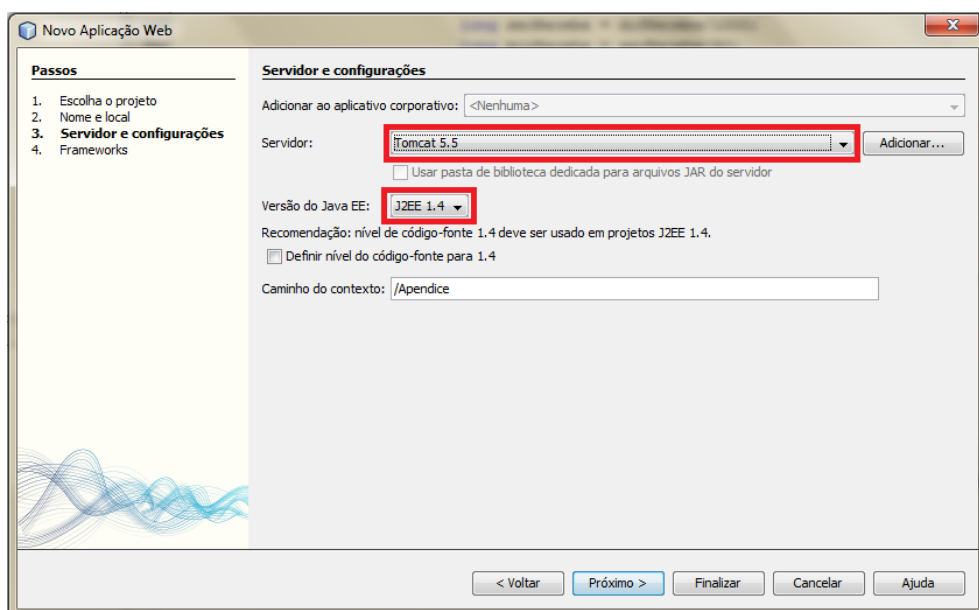


Figura B.3: Escolha do servidor e da versão do J2EE.

Realizados estes passos, caso não for necessário o uso de nenhum framework, pode-se clicar em "Finalizar". Concluído os passos de criação de um projeto Web é necessário criar um novo arquivo. A seguir, a Figura B.4 mostra qual tipo de arquivo deve ser criado.

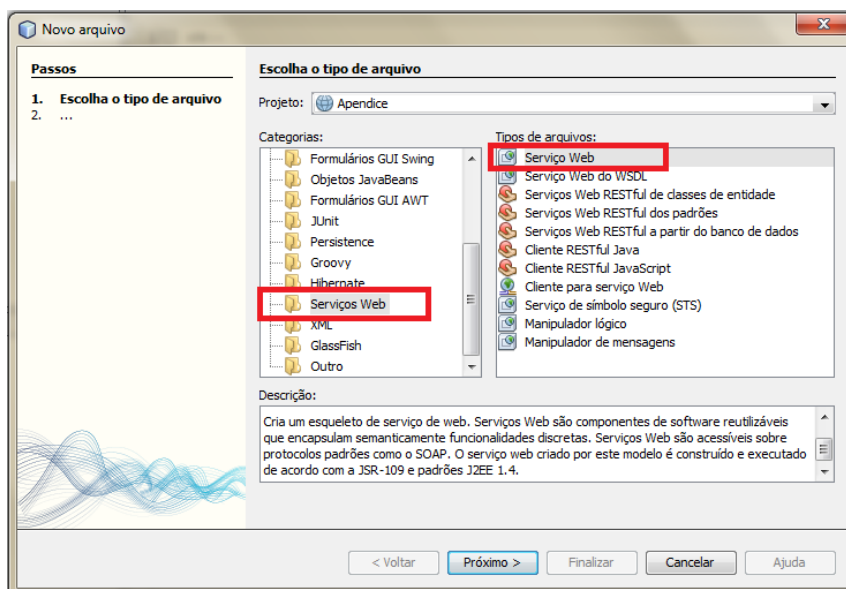


Figura B.4: Criação de um arquivo do tipo Serviço Web (*Web Service*).

A Figura B.5 representa a tela a qual se deve nomear o arquivo *Web service* e indicar o pacote de destino do mesmo.

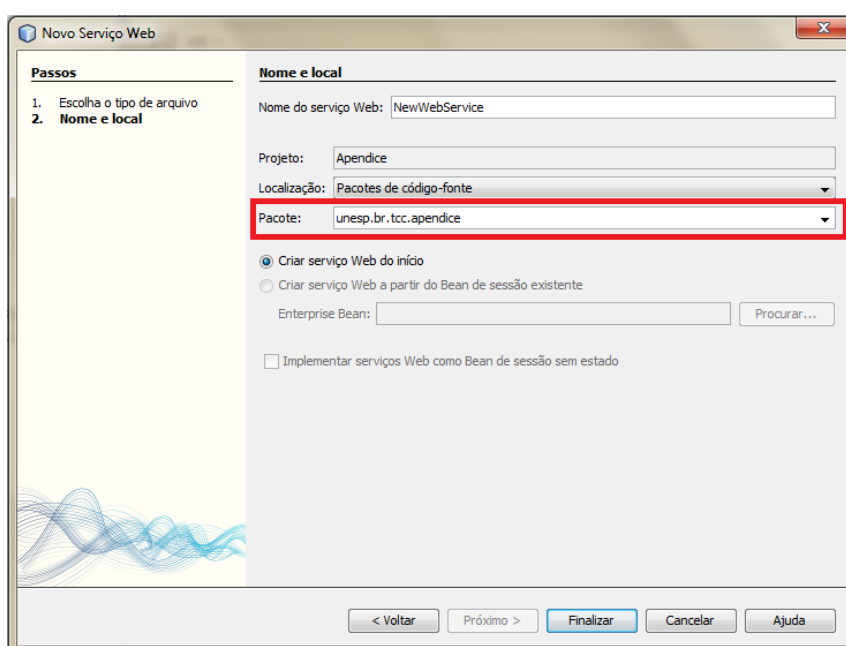


Figura B.5: Nome e pacote onde será armazenado o arquivo *Web Service*.

Seguidos os passos acima, uma nova classe deverá ser gerada no projeto, tal classe representa a classe de serviços que realizará a comunicação com o cliente. Depois de implementado as classes e métodos desejados, o projeto poderá ser implantado no servidor, onde fornecerá seus serviços aos clientes consumidores.

Apêndice C

Criação de um cliente para consumo de um *Web service* utilizando a IDE NetBeans 6.9.1

Para a geração de uma classe de consumo de um *Web service* em uma aplicação JAVA, na IDE NetBeans, será considerado que já existe uma aplicação criada.

Ao escolher a opção na IDE de novo arquivo, deve-se selecionar o tipo de arquivo conforme a Figura C.1.

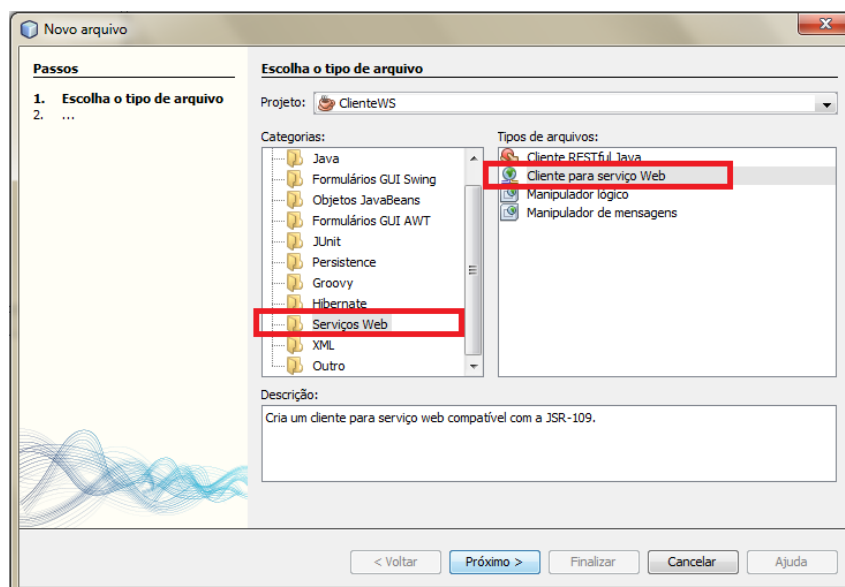


Figura C.1: Nova classe para consumo de um *Web Service*.

Ao clicar em "Próximo", a seguinte tela, apresentada pela Figura C.2 irá aparecer, a qual deve-se indicar o endereço do WSDL do *Web service* o qual se deseja gerar o cliente para consumo.

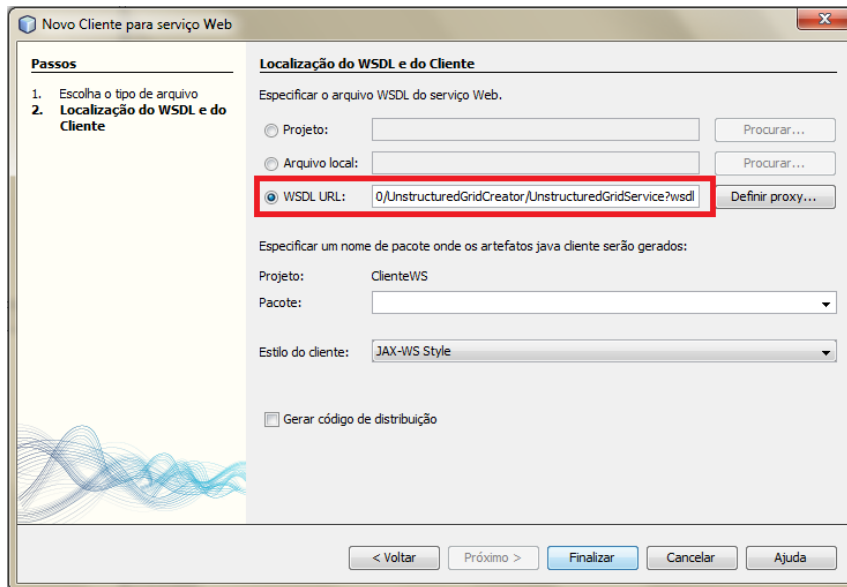


Figura C.2: Indicação do WSDL do *Web service* para consumo.

Ao finalizar tal processo, algumas classes e pacotes serão gerados automaticamente pela IDE NetBeans, disponibilizando assim, o consumo do *Web service* em questão através dessas classes geradas.

A Figura C.3 mostra alguns pacotes de classes gerados automaticamente pela ferramenta.

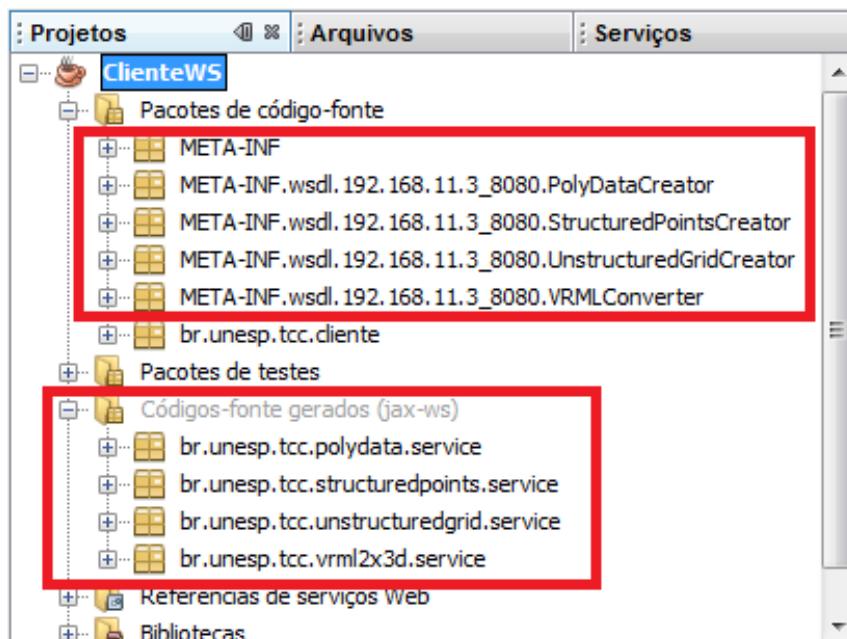


Figura C.3: Pacotes de classes gerados pela IDE NetBeans.

Apêndice D

WSDL do *Web service* PolyDataCreator

Segue abaixo o documento XML referente ao WSDL do *Web service* PolyDataCreator completo, acessado através do endereço <http://localhost:8080/PolyDataCreator/PolyDataService?wsdl>.

```
<?xml version='1.0' encoding='UTF-8'?><!--
Published by JAX-WS RI at http://jax-ws.dev.java.net.
RI's version is JAX-WS RI 2.2-hudson-740-.--><!--
Generated by JAX-WS RI at http://jax-ws.dev.java.net.
RI's version is JAX-WS RI 2.2-hudson-740-.
--><definitions xmlns:wsu="http://docs.oasis-open.org/wss/
2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://service.polydata.tcc.unesp.br/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://service.polydata.tcc.unesp.br/"
name="PolyDataServiceService">
<types>
<xsd:schema>
<xsd:import namespace="http://service.polydata.tcc.unesp.br/"
schemaLocation="http://localhost:8080/PolyDataCreator/
PolyDataService?xsd=1" />
</xsd:schema>
</types>
<message name="setBackgroundColor">
<part name="parameters" element="tns:setBackgroundColor" />
</message>
<message name="getVRML">
<part name="parameters" element="tns:getVRML" />
</message>
<message name="getVRMLResponse">
<part name="parameters" element="tns:getVRMLResponse" />
</message>
```

```

<message name="setInterpolation">
  <part name="parameters" element="tns:setInterpolation" />
</message>
<message name="createVRML">
  <part name="parameters" element="tns:createVRML" />
</message>
<message name="setOpacity">
  <part name="parameters" element="tns:setOpacity" />
</message>
<message name="setFile">
  <part name="parameters" element="tns:setFile" />
</message>
<portType name="PolyDataService">
  <operation name="setBackgroundcolor">
    <input wsam:Action="http://service.polydata.tcc.unesp.br/
PolyDataService/setBackgroundColor" message=
"tns:setBackgroundColor" />
  </operation>
  <operation name="getVRML">
    <input wsam:Action="http://service.polydata.tcc.unesp.br/
PolyDataService/getVRMLRequest" message="tns:getVRML" />
    <output wsam:Action="http://service.polydata.tcc.unesp.br/
PolyDataService/getVRMLResponse" message=
"tns:getVRMLResponse" />
  </operation>
  <operation name="setInterpolation">
    <input wsam:Action="http://service.polydata.tcc.unesp.br/
PolyDataService/setInterpolation" message=
"tns:setInterpolation" />
  </operation>
  <operation name="createVRML">
    <input wsam:Action="http://service.polydata.tcc.unesp.br/
PolyDataService/createVRML" message="tns:createVRML" />
  </operation>
  <operation name="setOpacity">
    <input wsam:Action="http://service.polydata.tcc.unesp.br/
PolyDataService/setOpacity" message="tns:setOpacity" />
  </operation>
  <operation name="setFile">
    <input wsam:Action="http://service.polydata.tcc.unesp.br/
PolyDataService/setFile" message="tns:setFile" />
  </operation>
</portType>
<binding name="PolyDataServicePortBinding" type=
"tns:PolyDataService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
  <operation name="setBackgroundcolor">
    <soap:operation soapAction="" />
  <input>

```



```

<soap:body use="literal" />
</input>
</operation>
<operation name="getVRML">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
<operation name="setInterpolation">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="createVRML">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="setOpacity">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="setFile">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
</binding>
<service name="PolyDataServiceService">
<port name="PolyDataServicePort" binding=
"tns:PolyDataServicePortBinding">
<soap:address location="http://localhost:8080/
PolyDataCreator/PolyDataService" />
</port>
</service>
</definitions>

```

Apêndice E

WSDL do *Web service* UnstructuredGridCreator

Segue abaixo o documento XML referente ao WSDL do *Web service* UnstructuredGridCreator completo, acessado através do endereço <http://localhost:8080/UnstructuredGridCreator/UnstructuredGridService?wsdl>.

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Published by JAX-WS RI at
http://jax-ws.dev.java.net. RI's version is JAX-WS RI
2.2-hudson-740-. --><!-- Generated by JAX-WS RI at
http://jax-ws.dev.java.net. RI's version is
JAX-WS RI 2.2-hudson-740-. --><definitions
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://service.unstructuredgrid.tcc.unesp.br/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://service.unstructuredgrid.tcc.unesp.br/"
name="UnstructuredGridServiceService">
<types>
<xsd:schema>
<xsd:import
namespace="http://service.unstructuredgrid.tcc.unesp.br/"
schemaLocation="http://192.168.11.3:8080/UnstructuredGridCreator/
UnstructuredGridService?xsd=1" />
</xsd:schema>
</types>
<message name="setBackgroundColor">
<part name="parameters" element="tns:setBackgroundColor" />
</message>
<message name="getVRML">
<part name="parameters" element="tns:getVRML" />
</message>
<message name="getVRMLResponse">
```

```

<part name="parameters" element="tns:getVRMLResponse" />
</message>
<message name="setInterpolation">
<part name="parameters" element="tns:setInterpolation" />
</message>
<message name="createVRML">
<part name="parameters" element="tns:createVRML" />
</message>
<message name="setOpacity">
<part name="parameters" element="tns:setOpacity" />
</message>
<message name="setFile">
<part name="parameters" element="tns:setFile" />
</message>
<portType name="UnstructuredGridService">
<operation name="setBackgroundcolor">
<input wsam:Action="http://service.unstructuredgrid.tcc.unesp.br/
UnstructuredGridService/setBackgroundColor"
message="tns:setBackgroundColor" />
</operation>
<operation name="getVRML">
<input wsam:Action="http://service.unstructuredgrid.tcc.unesp.br/
UnstructuredGridService/getVRMLRequest"
message="tns:getVRML" />
<output wsam:Action="http://service.unstructuredgrid.tcc.unesp.br/
UnstructuredGridService/getVRMLResponse"
message="tns:getVRMLResponse" />
</operation>
<operation name="setInterpolation">
<input wsam:Action="http://service.unstructuredgrid.tcc.unesp.br/
UnstructuredGridService/setInterpolation"
message="tns:setInterpolation" />
</operation>
<operation name="createVRML">
<input wsam:Action="http://service.unstructuredgrid.tcc.unesp.br/
UnstructuredGridService/createVRML" message="tns:createVRML" />
</operation>
<operation name="setOpacity">
<input wsam:Action="http://service.unstructuredgrid.tcc.unesp.br/
UnstructuredGridService/setOpacity" message="tns:setOpacity" />
</operation>
<operation name="setFile">
<input wsam:Action="http://service.unstructuredgrid.tcc.unesp.br/
UnstructuredGridService/setFile" message="tns:setFile" />
</operation>
</portType>
<binding name="UnstructuredGridServicePortBinding"
type="tns:UnstructuredGridService">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />

```

```

<operation name="setBackgroundColor">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="getVRML">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
<operation name="setInterpolation">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="createVRML">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="setOpacity">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="setFile">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
</binding>
<service name="UnstructuredGridServiceService">
<port name="UnstructuredGridServicePort"
binding="tns:UnstructuredGridServicePortBinding">
<soap:address location="http://192.168.11.3:8080/
UnstructuredGridCreator/UnstructuredGridService" />
</port>
</service>
</definitions>

```

Apêndice F

WSDL do *Web service* StructuredPointsCreator

Segue abaixo o documento XML referente ao WSDL do *Web service* StructuredPointsCreator completo, acessado através do endereço <http://localhost:8080/StructuredPointsCreator/StructuredPointsService?wsdl>.

```
<?xml version='1.0' encoding='UTF-8'?><!-- Published by
JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
JAX-WS RI 2.2-hudson-740-. --><!-- Generated by JAX-WS RI at
http://jax-ws.dev.java.net. RI's version is
AX-WS RI 2.2-hudson-740-. -->
<definitions xmlns:wsu="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://service.structuredpoints.tcc.unesp.br/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://service.structuredpoints.tcc.unesp.br/"
name="StructuredPointsServiceService">
<types>
<xsd:schema>
<xsd:import namespace="
http://service.structuredpoints.tcc.unesp.br/"
schemaLocation="http://192.168.11.3:8080/
StructuredPointsCreator/StructuredPointsService?xsd=1" />
</xsd:schema>
</types>
<message name="setBackgroundColor">
<part name="parameters" element="tns:setBackgroundColor" />
</message>
<message name="getVRML">
<part name="parameters" element="tns:getVRML" />
</message>
<message name="getVRMLResponse">
```

```

<part name="parameters" element="tns:getVRMLResponse" />
</message>
<message name="setInterpolation">
<part name="parameters" element="tns:setInterpolation" />
</message>
<message name="createVRML">
<part name="parameters" element="tns:createVRML" />
</message>
<message name="setOpacity">
<part name="parameters" element="tns:setOpacity" />
</message>
<message name="setFile">
<part name="parameters" element="tns:setFile" />
</message>
<portType name="StructuredPointsService">
<operation name="setBackground">
<input wsam:Action="http://service.structuredpoints.tcc.unesp.br/
StructuredPointsService/setBackgroundColor"
message="tns:setBackgroundColor" />
</operation>
<operation name="getVRML">
<input wsam:Action="http://service.structuredpoints.tcc.unesp.br/
StructuredPointsService/getVRMLRequest"
message="tns:getVRML" />
<output wsam:Action="http://service.structuredpoints.tcc.unesp.br/
StructuredPointsService/getVRMLResponse"
message="tns:getVRMLResponse" />
</operation>
<operation name="setInterpolation">
<input wsam:Action="http://service.structuredpoints.tcc.unesp.br
StructuredPointsService/setInterpolation"
message="tns:setInterpolation" />
</operation>
<operation name="createVRML">
<input wsam:Action="http://service.structuredpoints.tcc.unesp.br/
StructuredPointsService/createVRML" message="tns:createVRML" />
</operation>
<operation name="setOpacity">
<input wsam:Action="http://service.structuredpoints.tcc.unesp.br/
StructuredPointsService/setOpacity" message="tns:setOpacity" />
</operation>
<operation name="setFile">
<input wsam:Action="http://service.structuredpoints.tcc.unesp.br/
StructuredPointsService/setFile" message="tns:setFile" />
</operation>
</portType>
<binding name="StructuredPointsServicePortBinding"
type="tns:StructuredPointsService">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />

```

```

<operation name="setBackgroundColor">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="getVRML">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
<operation name="setInterpolation">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="createVRML">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="setOpacity">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
<operation name="setFile">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
</operation>
</binding>
<service name="StructuredPointsServiceService">
<port name="StructuredPointsServicePort "
binding="tns:StructuredPointsServicePortBinding">
<soap:address location="http://192.168.11.3:8080/
StructuredPointsCreator/StructuredPointsService" />
</port>
</service>
</definitions>

```

Apêndice G

WSDL do *Web service* VRMLConverter

Segue abaixo o documento XML referente ao WSDL do *Web service* VRMLConverter completo, acessado através do endereço <http://localhost:8080/VRMLConverter/Vrml2x3dService?wsdl>.

```
<?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI
at http://jax-ws.dev.java.net. RI's version is
JAX-WS RI 2.2-hudson-740-. --><!-- Generated by JAX-WS RI
at http://jax-ws.dev.java.net. RI's version is
JAX-WS RI 2.2-hudson-740-. -->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://service.vrml2x3d.tcc.unesp.br/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://service.vrml2x3d.tcc.unesp.br/"
name="Vrml2x3dServiceService">
<types>
<xsd:schema>
<xsd:import namespace="http://service.vrml2x3d.tcc.unesp.br/"
schemaLocation=
"http://192.168.11.3:8080/VRMLConverter/Vrml2x3dService?xsd=1" />
</xsd:schema>
</types>
<message name="getX3D">
<part name="parameters" element="tns:getX3D" />
</message>
<message name="getX3DResponse">
<part name="parameters" element="tns:getX3DResponse" />
</message>
<message name="setFile">
<part name="parameters" element="tns:setFile" />
</message>
```



```

<portType name="Vrml2x3dService">
  <operation name="getX3D">
    <input wsam:Action="http://service.vrml2x3d.tcc.unesp.br/
Vrml2x3dService/getX3DRequest" message="tns:getX3D" />
    <output wsam:Action="http://service.vrml2x3d.tcc.unesp.br/
Vrml2x3dService/getX3DResponse" message="tns:getX3DResponse" />
  </operation>
  <operation name="setFile">
    <input wsam:Action="http://service.vrml2x3d.tcc.unesp.br/
Vrml2x3dService/setFile" message="tns:setFile" />
  </operation>
</portType>
<binding name="Vrml2x3dServicePortBinding"
type="tns:Vrml2x3dService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
  <operation name="getX3D">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
  <operation name="setFile">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
  </operation>
</binding>
<service name="Vrml2x3dServiceService">
  <port name="Vrml2x3dServicePort"
binding="tns:Vrml2x3dServicePortBinding">
    <soap:address location="http://192.168.11.3:8080/
VRMLConverter/Vrml2x3dService" />
  </port>
</service>
</definitions>

```