UNIVERSIDADE DE CAXIAS DO SUL

Centro de Computação e Tecnologia da Informação

Curso de Bacharelado em Sistemas de Informação

Maycon Fábio de Oliveira

CLIENTE DE WEB SERVICE DINÂMICO COM INTERFACE EM JAVAFX

Caxias do Sul

2010

Maycon Fábio de Oliveira

CLIENTE DE WEB SERVICE DINÂMICO COM INTERFACE EM JAVAFX

Trabalho de conclusão do curso de Bacharelado em Sistemas de Informação realizado na Universidade de Caxias do Sul.

Alexandre E. Krohn Nascimento
Orientador

Caxias do Sul
2009

“They have computers, and they may have other weapons of mass destruction”.

— Janet Reno, Attorney General, February 2, 1998

AGRADECIMENTOS

Agradeço a minha família e a minha namorada Vanessa Zattera por acreditarem no meu trabalho e me apoiarem na vida acadêmica e profissional.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Sigla	Significado em Inglês	Significado em Português
API	*Application Programming Interface*	Interface de Programação de Aplicativos
BPM	*Business Process Modeling*	Modelagem de Processos de Negócio
CORBA	*Common Object Request Broker Architecture*	Arquitetura intermediadora entre requisições de objetos comuns
DCE	*Distributed Computing Enviroment*	Ambiente de computação distribuida
DTD	*Document Type Definition*	Documento de Definição de Tipos
EDI	*Eletronic Data Interchange*	Transferência Eletrônica de Dados
ERP	*Enterprise Resource Planning*	Planejamento de Recursos Empresariais
ESB	*Enterprise Service Bus*	Barramento de Serviços Corporativos
HTTP	*Hipertext Transfer Protocol*	Protocolo de Transferência de Hipertexto
IANA	*Internet Assigned Numbers Authority*	Autoridade de nomeação de números para a Internet
JAR	*Java Application Resource*	Recurso de Aplicação Java
JAX-WS	*Java API for XML Web Services*	API Java para Web Services baseados em XML
JDK	*Java Development Kit*	Kit de Desenvolvimento Java
JVM	*Java Virtual Machine*	Máquina virtual Java
NFE		Nota Fiscal Eletrônica
POJO	*Plain Old Java Object*	Velho e Simples Objeto Java
RFC	*Request for Comments*	Pedido para comentários
RIA	*Rich Internet Application*	Aplicações Ricas para Internet
RMI	*Remote Method Invocation*	Invocação de Métodos Remotos
RPI	*Remote Procedure Invocation*	Invocação de Procedimentos

		Remotos
SGBD		Sistema de Gerenciamento de Banco de Dados
SGML	*Standard Generalized Markup Language*	Linguagem de marcação generalizada padrão
SMTP	*Simple Mail Transfer Protocol*	Protocolo Simples para Transferência de Emails
SOA	*Service Oriented Architecture*	Arquitetura Orientada a Serviços
SOAP	*Simple Object Access Protocol*	Protocolo de Acesso simples ao objeto
SQL	*Structured Query Language*	Linguagem de Pesquisa Estruturada
TISS		*Troca de Infomações em Saúde Suplementar*
UDDI	*Universal Description, Discovery and Integration*	Descrição, Descobrimento e Integração Universal
URI	*Uniform Resouce Identifiers*	Identificador de Recursos Uniforme
URL	*Uniform Resource Locators*	Localizador de Recursos Uniforme
UTF	*Unicode Transformation Format*	Formato de Transformação Unicode
W3C	*World Wide Web Consortium*	Consórcio Gerenciador da Web
WS-I	*Web Service Interoperability Organization*	Organização para interoperabilidade de Web Services
XML	*Extensible Markup Language*	Linguagem de marcação extensiva
WSDL	*Web Service Description Language*	Linguagem de Descrição de Web Service

RESUMO

Este trabalho apresenta a análise e implementação de um cliente de *Web Service* dinâmico, que utiliza a tecnologia JavaFX para a camada de apresentação. O software tem como objetivo auxiliar o usuário na interação com diversos serviços, sem a necessidade da construção de programas específicos para este fim e sem precisar de grande conhecimento para realizar esta operação.

Palavras-chaves: Integração de sistemas, SOA, XML, *Web Services*, JavaFX

ABSTRACT

This work presents the analysis and implementation of a Dynamic Web Service client which uses the technology JavaFX for the presentation layer. The software has objective to aid the user in the interaction with various services, without the need of construct of the specific programs for this purpose and without great knowledge to execute this operation.

Keywords: System Integration, SOA, XML, Web Services, JavaFX

SUMÁRIO

1 INTRODUÇÃO

A falta de padrões abertos para troca de informações entre sistemas de informações foi, durante um passado bastante recente, um fator dificultador para organizações que tentavam manter seus dados unificados. Nesta época, era necessário que clientes e servidores de aplicações para a Internet fossem implementados a partir de acertos prévios para que informações fluíssem entre os agentes (NETO, 2006). Protocolos proprietários e formatos de mensagens combinados durante o tempo de desenvolvimento da solução podiam não ser a opção mais correta, mas para a demanda em épocas passadas era mais que o suficiente para ocorrer com sucesso uma integração.

Para permitir a interação entre softwares de diferentes plataformas, uma solução encontrada foi o desenvolvimento de interfaces por parte dos fabricantes, mas como eram soluções proprietárias, além de caras (para a programação e para operação), não eram flexíveis como as necessidades das organizações passaram a ser a cada dia (CRUZ, 2008).

Passados alguns anos e pelo aumento no volume dessas necessidades, o padrão SOA (*Service Oriented Architeture*) surgiu como uma opção bastante atrativa, por facilitar a integração de sistemas, cortando custos e agilizando processos (GANDOLPHO, 2010).

Com o surgimento de padrões abertos como o SOA, a necessidade de contratação de empresas específicas para concretizar projetos de integração de sistemas, passou a ser opcional, uma vez que informações sobre os padrões estão disponíveis para qualquer um que se interesse em acompanhá-las (uma vez que elas estão constantemente evoluindo), possibilitando o surgimento de softwares com diversas naturezas de comercialização, aumentando assim as alternativas do consumidor.

Um dos padrões que surgiu e que está se mostrando altamente utilizado principalmente por influência de órgãos governamentais como os Correios é a utilização de *Web Services* para integração de sistemas (TERZIAN, 2010). Outros órgãos governamentais adotaram o padrão, tais como a SEFAZ (Secretaria da Fazenda) com o projeto da NFE (Nota Fiscal Eletrônica) (SEFAZ, 2010) e a ANS (Agenda Nacional de Saúde) com o projeto TISS (Troca de Informação em Saúde Suplementar) (ANS, 2010) .

A W3C (*World Wide Web Consortium*) define *Web Services* como um “sistema de software projetado para suportar a interoperabilidade entre máquinas em cima de uma rede. O mesmo provê uma interface descrita em um formato processável por máquinas”. Segundo este padrão, é possível extrair informações de sua interface a ponto de entender o que é necessário

para conseguir se comunicar através do mesmo, e com isso iniciar uma conversação com o sistema destino.

Uma vez desenvolvido, um *Web Service* não deve mudar suas definições e parâmetros para acesso, pois diversos clientes podem estar acessando, podendo assim provocar falhas nas aplicações dos mesmos (NETO, 2006). Por este motivo, programas que acessam *Web Services* são construídos de forma fixa, ou seja, impossibilitando comunicação com outros sistemas através do mesmo programa.

Porque não construir dinamicamente o programa que realizará a comunicação com estes sistemas? Construir dinamicamente significa mapear todas as situações possíveis de ocorrer dentro de uma interface de *Web Service*, para que o programa seja capaz de se adaptar as mesmas, criando as interfaces gráficas necessárias para entrada de dados por parte do usuário. Programas dinâmicos, devido a essa necessidade de prever as situações possíveis, são geralmente mais complicados de serem construídos, mas não impossíveis.

O objetivo deste trabalho é a construção de um software capaz de dinamicamente invocar um serviço, possibilitando que o usuário entre com os dados desejados a serem submetidos, através de uma interface gráfica gerada automaticamente a partir da interpretação das características do serviço.

Para atingir o objetivo proposto, este trabalho está organizado da seguinte forma:

No capítulo 2 são apresentadas as evoluções das metodologias para integração de sistemas até chegarem na arquitetura SOA e sua implementação *Web Services*.

No capítulo 3 são apresentadas as tecnologias envolvidas na integração de sistemas e também na interface que será apresentada ao usuário para entrada de dados.

O capítulo 4 contempla a explicação detalhada do projeto desenvolvido, utilizando-se de diagramas e figuras para facilitar o entendimento.

O capítulo 5 apresenta uma série de considerações sobre o projeto e as situações ocorridas durante o desenvolvimento do mesmo.

O capítulo 6 realiza as considerações finais do projeto, apresentando sugestões para continuidade do mesmo.

2 INTEGRAÇÃO DE SISTEMAS

Processos interligados e respostas imediatas são, sem dúvida, o sonho de todos os empresários em relação à forma como sua empresa deve proceder e ser administrada. Hoje, estes itens, principalmente em grandes organizações, são resultado de uma razoável utilização da informática como ferramenta para otimizar os processos tanto internos como externos. Como conseqüência, temos um aumento da dependência da organização em sistemas de informações computadorizados, que podem ter sido construídos em diversos momentos, utilizando as mais variadas tecnologias disponíveis, desta forma, inviabilizando uma comunicação nativa entre os sistemas.

LINTHICUM (1999) confirma este fato ao afirmar que "como a dependência das empresas em tecnologia se tornou mais complexa e maior, a necessidade de um método de integração de aplicações diferentes em um conjunto unificado de processos de negócios tem surgido como uma prioridade".

Hoje, não é difícil encontrarmos empresas com uma grande variedade de soluções, sejam elas desenvolvidas internamente, terceirizadas ou simplesmente fazendo parte do "legado", criando assim uma situação que dificulta drasticamente seu controle.

Uma série de padrões podem estar envolvidos em uma integração de sistemas, possivelmente estando diretamente interligadas para completar o processo. HOHPE (2003), define estes padrões em 4 tipos distintos:

- Transferência de arquivos;
- Banco de dados compartilhado;
- Envio de mensagens;
- Invocação de procedimentos remotos.

Na próxima seção cada um destes padrões será melhor explicado.

2.1 Transferência de arquivos

Para HOHPE (2003), "Arquivos são um mecanismo de armazenamento universal, embutido em qualquer sistema operacional, disponível em qualquer linguagem". Essa

natureza simples de operação dos arquivos transformou o modelo de integração via transferência de arquivo, no mecanismo mais utilizado para integração de sistemas.

Este modelo consiste no transmissor persistindo informações que o receptor espera receber dentro de um arquivo, e por sua vez o receptor importa este arquivo para seu sistema, conforme a Figura 1.

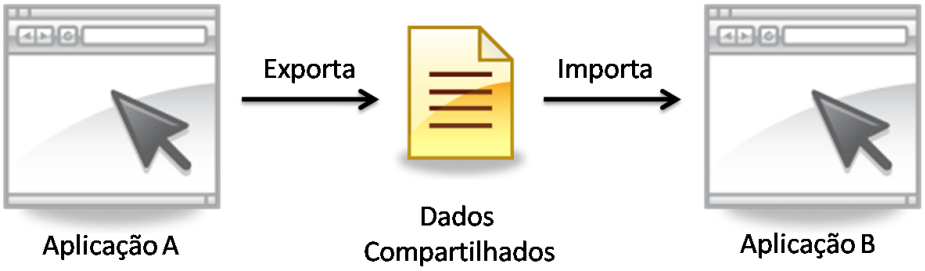


Figura 1 - Transferência de arquivos

Transferência de arquivos também é um modelo de integração com baixo acoplamento, ou seja, as aplicações envolvidas em uma integração não necessitam uma das outras para seu funcionamento, simplesmente as mesmas trocam mensagens para compartilhamento de dados.

Porém, dois grandes problemas existem neste modelo. O primeiro reside na necessidade da prévia negociação do formato que será utilizado para controle das informações dentro do arquivo. O segundo problema é relativo a freqüência de leitura dos arquivos do sistema receptor. Devido ao atraso na transferência e importação dos arquivos entre os sistemas, pode ocorrer uma dessincronia dos dados, ocasionando a inexistência temporária de registros no sistema receptor. Também é necessário a utilização de controles adicionais sobre a origem dos dados (sistema emissor), para evitar chaves duplicadas no sistema de destino, uma vez que as faixas de valores importados podem coincidir com os já existentes.

2.2 Banco de dados compartilhado

Este modelo consiste na centralização da informação dentro de um único repositório de dados visível para todas as aplicações, conforme a Figura 2 demonstra. Este modelo resolve alguns dos problemas existentes na transferência de arquivos, principalmente no quesito velocidade de atualização e disponibilidade dos dados.

Outra grande vantagem deste modelo é o formato dos dados. Segundo HOHPE (2009), a maior parte das plataformas de desenvolvimento suportam a linguagem SQL (*Structured Query Language*) que é utilizada para manipulação de dados dentro de um banco de dados, eximindo assim a responsabilidade do desenvolvedor em previamente combinar o formato de transferência de dados.

Contudo, o compartilhamento de banco de dados necessita de atenção. Múltiplas aplicações acessando o mesmo banco de dados podem resultar em bloqueios, ou seja, em registros sendo bloqueados, impedindo acesso de outras aplicações que desejam manipular a mesma informação. Além disto, devido às diferentes implementações de SGBD (Sistemas de gerenciamento de banco de dados) a sintaxe da linguagem SQL pode variar, podendo resultar em problemas de portabilidade de uma aplicação a qual desejamos conectar ao mesmo banco de dados compartilhado.

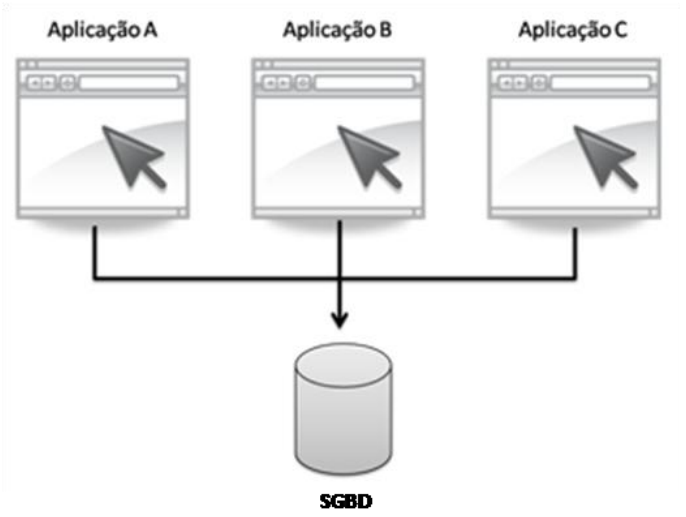


Figura 2 - Compartilhamento de banco de dados

2.3 Envio de Mensagens

Algumas vezes o requisito é integrar aplicações heterogêneas com o mínimo acoplamento, a melhor performance possível e para múltiplos destinatários. Nos modelos anteriores foi visto que a melhor opção para evitar acoplamento entre aplicações é utilizando transferência de arquivos, mas a custo da existência de um atraso entre o dado ser enviado e

ser processado. Para atender esta situação, o modelo de envio de mensagens é utilizado como alternativa.

O envio de mensagens consiste na transmissão de pequenas mensagens até o sistema de mensagens, aonde o mesmo é responsável por avisar as aplicações conectadas que existem novas mensagens para serem processadas. Este modelo originalmente utiliza a forma de envio assíncrona, ou seja, a mensagem é enviada, mas a aplicação emissora não fica aguardando a resposta.

Os canais de mensagens são endereços lógicos no sistema de mensagens. Como eles são implementados depende do produto de envio de mensagens utilizado.

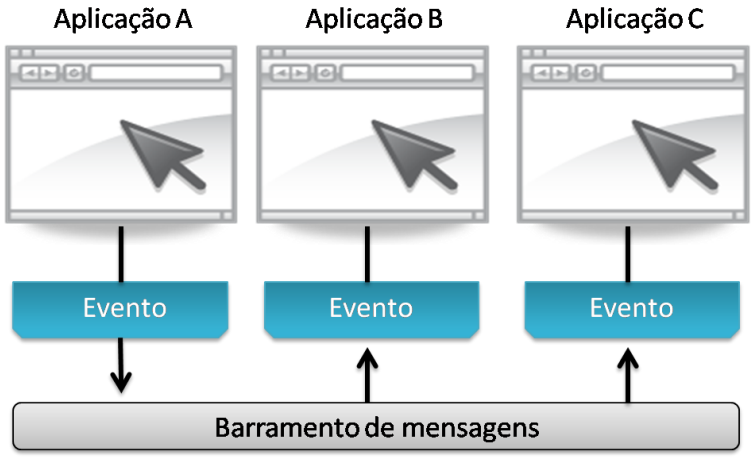


Figura 3 - Transferência de mensagens

Como exemplo de implementações de sistema de mensagens pode-se citar JMS (*Java Messaging System*) e ESB (*Enterprise Service Bus*).

2.4 Invocação de Procedimentos Remotos

RPI (*Remote Procedure Invocation*) ou invocação de procedimentos remotos aplica o princípio de encapsulamento para aplicações de integração (HOHPE, 2003). Se uma aplicação necessita manipular alguma informação da qual outra aplicação é proprietária, uma chamada direta a aplicação destino é realizada, conforme a Figura 4 demonstra. A integridade dos dados a serem manipulados é responsabilidade da aplicação que esta sendo invocada.

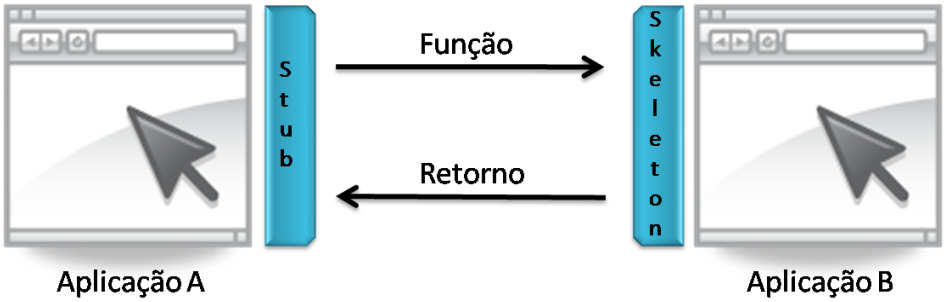


Figura 4 - Invocação de procedimentos remotos

Em linguagens como Java, RPI traz a vantagem de evitar a utilização de outras linguagens ou padrões para comunicação além dos próprios oferecidos pela própria linguagem. Isto facilita principalmente diminuindo a curva de aprendizagem do desenvolvedor.

Chamadas RPI são síncronas, ou seja, o emissor fica aguardando a resposta da solicitação até o seu processamento e retorno. O tempo de espera da resposta pode variar, pois como a invocação é remota, a performance estará diretamente relacionada a velocidade da rede. Por isso, a quantidade de chamadas deve ser sempre verificada a fim de evitar o tráfego desnecessário de informação na rede.

Alguns exemplos de implementações deste modelo são: CORBA (*Common Object Request Broker Architecture*), Java RMI (*Remote Method Invocation*), DCE (*Distributed Computing Enviroment*), *Web Services* entre outros.

Cada tecnologia tem suas características de formato para transmissão de dados e protocolos de comunicação, conforme a Tabela 1 demonstra.

Tabela 1 - Formatos e protocolos de modelos RPI. Fonte MARTINS (2005).

	Java RMI	CORBA	DCE	Web Services
Invocação de procedimentos ou serviços	Java RMI	CORBA	RPC	JAX-RPC, .NET
Formato da informação	Java Serializado	Representação comum de dados	Representação de dados na rede	XML
Transporte da informação	Stream	General Inter-ORB Protocol	Protocol Data Units	SOAP
Protocolo de comunicação	Java Remote Method Protocol	Internet Inter-ORB Protocol	RPC Connection Oriented Protocol (ex:TCP)	HTTP, SMTP, FTP
Descrição do serviço	Java Interface	CORBA IDL	DCE IDL	WSDL
Descoberta do serviço	Java Registry	Corba Object Service Naming	Cell Directory Service	UDDI

Na comparação da Tabela 1 também é possível verificar que com a exceção de *Web Services*, os outros protocolos usam mecanismos próprios para manipulação de seus serviços. Isto introduz o problema da falta de interoperabilidade entre aplicações criadas em tecnologias distintas. Uma vez que para *Web Services* os formatos exigidos são padrões abertos e os protocolos de comunicação necessários estão presentes na maior parte das linguagens de programação, o processo de integração entre sistemas ganha uma nova perspectiva, drasticamente eliminando o problema de falta de interoperabilidade.

RPI também é a base para a arquitetura de desenvolvimento de aplicações orientadas a serviço – SOA (*Service Oriented Architecure*) e é dentro desta arquitetura que a tecnologia *Web Services* utilizada neste trabalho, se encontra.

Vimos até aqui os modelos de integração existentes até chegar no modelo RPI, utilizado pela tecnologia de *Web Services*. No próximo capítulo esta tecnologia será, abordada de forma aprofundada, detalhando artefatos e outras tecnologias relacionadas.

3 TECNOLOGIAS

Algumas tecnologias estão presentes neste trabalho com o intuito de realizar uma interação completa entre o usuário e um serviço destino. Abaixo estão relacionadas as principais, sendo que algumas tais como WSDL (*Web Service Description Language*), XML (*Extensible Markup Language*) Schema e *Web Service* estão do lado servidor, JavaFX do lado do cliente e XML e SOAP (*Simple Object Access Protocol*) em ambos os lados como forma de comunicação entre os dois mundos.

3.1 Web Services

Segundo CRUZ (2008), "O surgimento do SOA deu-se pela introdução das tecnologias Web nas organizações. Em outras palavras, as tecnologias que antes eram usadas apenas na Internet passaram a ser usadas em sistemas de informações e em outras aplicações internas das organizações". O fator tecnológico que possibilitou a utilização de SOA nas organizações foi claramente citado por CRUZ (2008), mas os fatores motivacionais, tais como, reutilização e interoperabilidade são conceitos antigos no desenvolvimento de softwares e para a arquitetura SOA estes dois itens são seus pontos fortes.

Web Services, tecnologia que nasceu sob a arquitetura SOA, são definidos pela W3C (2010) como um "sistema de software projetado para suportar a interoperabilidade entre máquinas conectadas a uma rede. O mesmo provê uma interface descrita em um formato processável por máquinas". Este sistema de software ganhou popularidade devido a seus formatos de transmissão e descrição de dados baseados em padrões abertos tais como o XML (*Extensible Markup Language*), *XML Schema*, SOAP e WSDL descritos detalhadamente nos subitens que seguem.

3.1.1 XML

A W3C (2010) define XML como "um formato de texto simples e bastante flexível derivado do SGML (Standard *Generalized Markup Language*)."

Esta flexibilidade descrita pela W3C é alcançada graças a existência de unidades menores que podem ser reconhecidas e quando necessário, processadas individualmente. Estas unidades são chamadas de elementos e deve existir ao menos um elemento em um documento XML para que o mesmo seja válido (BRADLEY, 2002).

Devido a sua característica flexível, que possibilita que mensagens possam ser estendidas sem desestruturar o documento, o XML se tornou base para diversos tipos de tecnologias tanto para transmissão de dados como para aplicações que o utilizam como uma metalinguagem para armazenar informações do sistema ou até mesmo preferências de usuário.

Documentos XML podem compor estruturas hierárquicas e até mesmo ter elementos em recursividade, ou seja, um elemento pode diretamente ou indiretamente conter outras instâncias do mesmo tipo (BRADLEY, 2002). No exemplo da Figura 5, a estrutura hierárquica característica de um XML pode ser visualizada. Para uma estrutura recursiva, pode-se acrescentar a idéia de que um acessório de um produto pode ser composto de outros acessórios, como por exemplo, o carregador, descrito na linha 6, poderia ser composto por outros acessórios como adaptadores para diversos tipo de tomada.

Outra característica a ser visualizada na Figura 5 é que o XML também é uma linguagem "*human-readable*", ou seja, legível por humanos. Isto também contribuiu para o sucesso da linguagem frente aos outros padrões. Segundo LINTINCUM (1999), "XML é simples de entender e usar. XML pode trabalhar com grandes pedaços de informação e os consolidar em um documento XML – pedaços significantes que provêem estrutura e organização à informação."

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <produto>
3        <codigo>1000</codigo>
4        <descricao>Celular Nokia 5800</descricao>
5        <acessorios>
6              <acessorio>Carregador</acessorio>
7              <acessorio>Fone de ouvido</acessorio>
8        </acessorios>
9  </produto>
```

Figura 5 - Exemplo de arquivo XML

A linha 1 da Figura 5 contém a marcação que informa que o documento apresentado é do tipo XML em sua versão atual (1.0) e codificado com caracteres *unicode* do padrão UTF-8 (*8-bit Unicode Transformation Format*) .

Nas linhas que se seguem, pode-se verificar a declaração de um produto com seus devidos atributos como código, descrição e acessórios relacionados ao mesmo.

O XML serve como sintaxe de desenvolvimento das próximas tecnologias que serão apresentadas nos próximos itens.

3.1.2 XML Schema

A W3C (2010), define *XML Schema* como uma metalinguagem em XML utilizada para descrever a estrutura, conteúdo e a semântica de documentos XML. Esta tecnologia que reconstrói e estende consideravelmente a tecnologia DTD (*Document Type Definition*), foi apresentada pela W3C em maio de 2001 (W3C, 2010) e apresenta uma série de funcionalidades adequadas para sua utilização junto à interface de *Web Services*.

Observe um exemplo de XML Schema e na seqüência a explicação de cada elemento na Figura 6:

```
1  <?xml version="1.0"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:element name="cliente">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element name="nome" type="xs:string" />
7          <xs:element name="rua" type="xs:string" />
8          <xs:element name="bairro" type="xs:string" />
9          <xs:element name="CEP" type="xs:string" />
10       </xs:sequence>
11     </xs:complexType>
12   </xs:element>
13 </xs:schema>
```

Figura 6 - Exemplo de XML Schema

- **Declaração de XML (Linha 1):** Este elemento que também existe em XML simples, declara qual a versão do documento que estamos trabalhando. Por *XML Schema* ser baseado em marcações XML, o mesmo também herda esta marcação.
- **Início da seção schema (Linha 2):** Esta é a *tag* raiz que define que o documento XML não é um documento simples e sim um arquivo de validação de estruturas de documentos XML.
- **Definição de um elemento (Linha 3):** Um documento XML é composto por uma série de *tags*, que na declaração XML Schema são vistas como elementos. No exemplo é declarado o elemento "cliente" que será o elemento raiz da mensagem a ser enviada, neste caso uma mensagem de envio dos dados cadastrais de um cliente. Note que o elemento "cliente" não tem nenhum tipo de dado associado diretamente ao mesmo, uma vez que ele é um dado composto, ou seja, é formado por outros tipos de dados. Uma *tag* do tipo elemento pode se referir a outros tipos de dados do tipo simples ou até mesmo do tipo complexo, não tendo necessariamente que conter a composição da *tag* dentro de si mesmo. Para que isto seja possível basta informar o atributo "type" indicando qual tipo de dado que deseja-se que o mesmo faça referência.
- **Declaração de tipo de dados (Linha 4):** Através desta declaração informa-se que o conteúdo do tipo de dados cliente, é um tipo complexo, ou seja, composta por mais de um tipo de informação. A declaração de tipo complexo ou até mesmo de um tipo simples pode ser externalizada em outra seção caso deseje-se que a composição da *tag* possa ser reutilizada, neste caso por *tags* do tipo *element*.
- **Declaração da ordem dos atributos (Linha 5):** O elemento *sequence* informa que os atributos que virão abaixo são esperados na mesma ordem em que foram declarados. Isto é particularmente interessante quando se trabalha com *parsers* (interpretadores) XML que precisam verificar a estrutura do documento, pois é mais eficiente para o *parser* saber o que deve esperar na ordem correta ao invés de analisar todas as *tags* e para saber se alguma está faltando.

- **Declaração dos elementos do XML (Linha 6 até 9):** Aqui é onde realmente é informado quais são as *tags* que devem compor a mensagem. Note que também é declarado uma *tag* "element", assim como foi realizado com a *tag* principal "cliente". A diferença é que junto ao nome da *tag* objetivo (nome, rua, bairro e CEP), informa-se também o tipo de dados de cada uma destas, no caso String. Este tipos de dados são padrão na implementação *XML Schema*, assim como todos os relacionados na Figura 7.

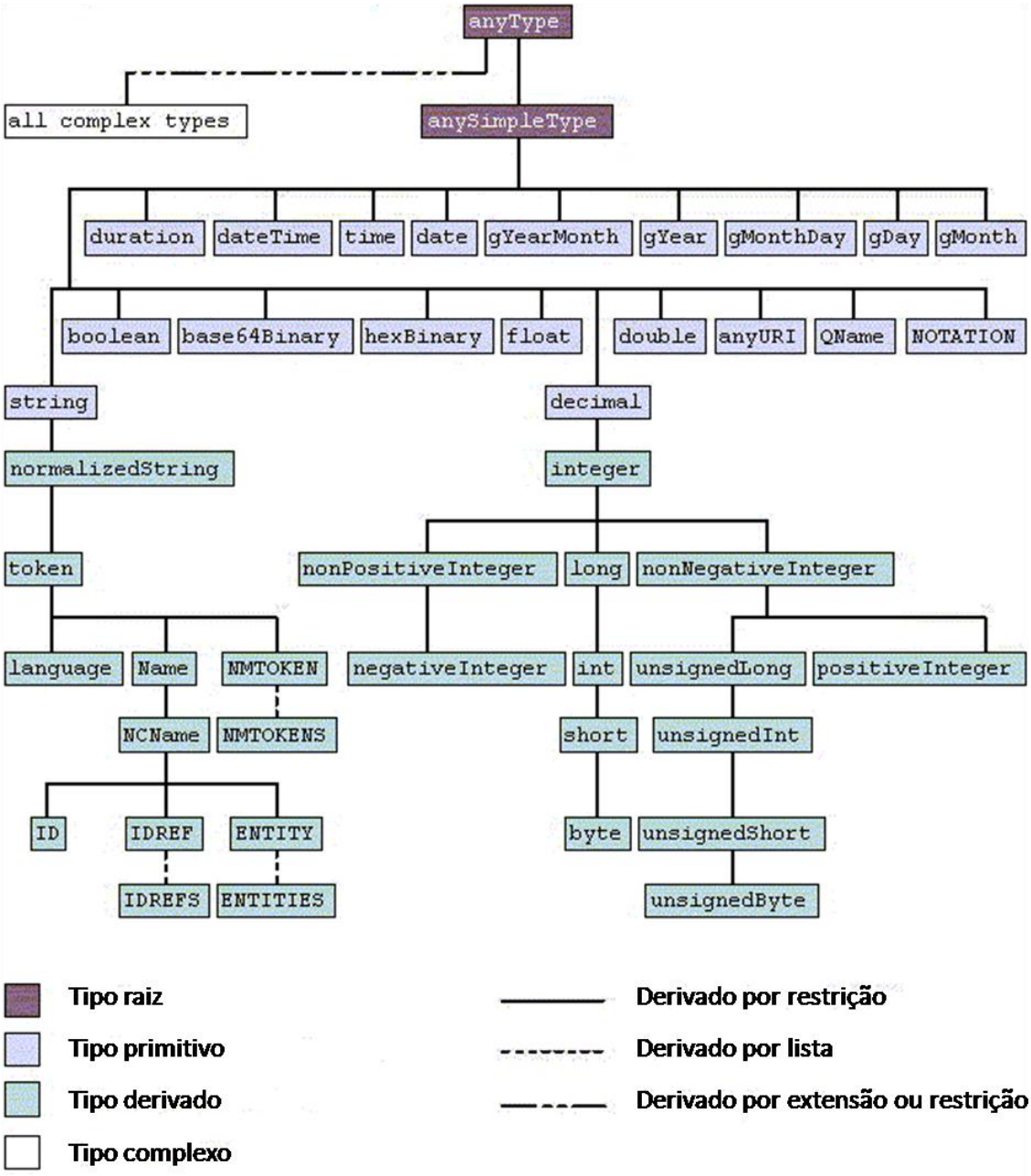


Figura 7 - Tipos de dados definido pelo XML Schema (W3C, 2010)

A árvore da Figura 7 demonstra toda a estrutura de tipos de dados suportados pelo *XML Schema*. A partir de testes realizados em APIs e em ferramentas de geração de código neutras (não distribuídas por um fabricante que tenha uma solução em *Web Services*) tais como Eclipse, NetBeans e Apache AXIS, foi verificado que os tipos de dados não presentes nesta árvore (portanto não oficiais W3C) não são suportados nativamente, necessitando que sejam importadas bibliotecas que acrescentem os devidos tipos de dados.

Outras características tornam a linguagem poderosa, tais como possibilidade de importar outros *XML Schemas*, gerenciamento de conflitos de nomes de tags, agrupadores, entre outros. Grande parte destas funcionalidades estará presente no ambiente de testes do software a ser desenvolvido neste trabalho.

O *XML Schema* tem uma participação importante na tecnologia *Web Services*. O mesmo está diretamente interligado a linguagem descritora de sua interface, o WSDL, que será visto no item 3.1.4. O *XML Schema* também define como é formada a mensagem que irá ser transportada dentro da mensagem SOAP, conforme o próximo item irá abordar.

3.1.3 SOAP

A interoperabilidade que é uma das principais características da tecnologia *Web Services*, é alcançada graças ao emprego de protocolos abertos e altamente difundidos como o SOAP.

Desenvolvido pela W3C, SOAP é definido pela mesma como um protocolo simples e leve baseado em XML para troca de informações estruturadas entre sistemas heterogêneos em um ambiente de computação distribuída (W3C, 2010).

Para permitir essa troca de informações entre sistemas heterogêneos (quanto a linguagem, versões, etc), uma mensagem SOAP é constituída de alguns elementos, conforme a Figura 8 demonstra. Veja que nesta Figura a mensagem foi encapsulada dentro do protocolo HTTP (*Hipertext Transfer Protocol*), sendo esta a forma mais tradicional de envio de mensagens para *Web Services*.

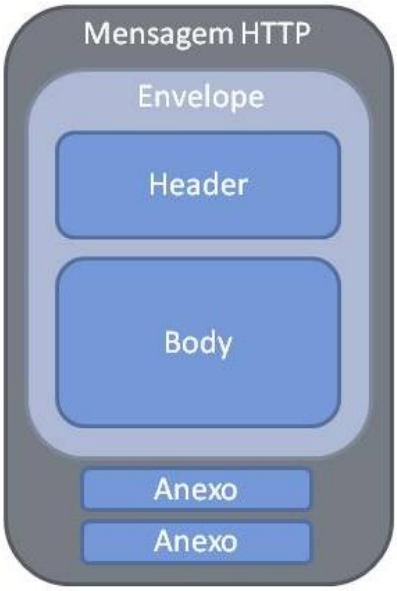


Figura 8 - Mensagem SOAP dentro do protocolo HTTP. (NETO, 2006)

A Figura 8 traduz graficamente como uma mensagem SOAP é estruturada. Na Figura 9 é apresentado como a mensagem é realmente apresentada durante uma transmissão via método POST. Repare que antes da mensagem SOAP iniciar, o cabeçalho HTTP também está descrito.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type:
text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header>
            <t:Transaction xmlns:t="some-URI"
                  SOAP-ENV:mustUnderstand="1">
                  5
            </t:Transaction>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
            <m:GetLastTradePrice xmlns:m="Some-URI">
                  <symbol>DEF</symbol>
            </m:GetLastTradePrice>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 9 - Exemplo de mensagem SOAP

Cada elemento de uma mensagem SOAP tem sua funcionalidade bem definida, de forma a permitir a interpretação da mensagem de forma padronizada, ou seja, a informação procurada sempre estará no local esperado. A seguir são relacionados os principais elementos, baseado nas explicações de ENGLANDER (2002).

- **Envelope**: Este é o maior elemento da mensagem SOAP e também a primeira marcação XML de uma transmissão. Esta *tag* deve ter a declaração do *namespace*[1] "http://schemas.xmlsoap.org/soap/envelope", para que a mesma represente um envelope SOAP válido. Todos os sub-elementos desta tag devem ser qualificados pelo prefixo definido a este *namespace*, que no caso do exemplo acima é "SOAP-ENV".

- **Header**: Esta é uma *tag* opcional dentro de uma mensagem SOAP e, se estiver presente, deve ser o primeiro elemento declarado após a declaração do envelope. Esta *tag* é principalmente utilizada para controle da mensagem que irá se seguir. Funções como autenticação, regras de roteamento e transação podem ser acrescentadas nesta *tag*.

- **Body:** Esta é a principal *tag* da mensagem SOAP e por isto sua presença é obrigatória. Nesta *tag* estará presente a mensagem de requisição ou de resposta que desejamos trafegar.

SOAP é um padrão que está em constante evolução. Em junho de 2003, a W3C apresentou a especificação do SOAP 1.2 como forma de melhorar a interoperabilidade entre *Web Services* e ampliar as funcionalidades do SOAP 1.1 (W3C, 2010). Esta informação é de grande importância para um software que deseja estabelecer comunicação com SOAP, uma vez que uma mensagem SOAP deve ser escrita no modelo a qual o servidor espera receber, caso contrário a mensagem pode ser negada.

Algumas das diferenças entre os dois modelos são: alteração de sintaxe de atributos, substituição do atributo SOAPAction pelo código HTTP 427 registrado na IANA (*Internet Assigned Numbers Authority*), registro do tipo de dado "application/soap+xml" através da

[1] Qualificador de nomes de elementos e atributos usados em XML, associando-os a referencias de URIs (*Uniform Resource Identifier*) (W3C, 2010).

RFC (*Request For Comments*) 3902 entre outros. A lista completa de alterações pode ser encontrada no site da W3C.

Uma mensagem SOAP está descrita dentro do documento descritor de serviço, o WSDL, que será melhor detalhado na seção a seguir.

3.1.4 WSDL

Conforme NETO (2006), o WSDL tem a função de descrever *Web Services* de modo a preencher lacunas principais como: informar ao consumidor o que o serviço executa, como o consumidor pode invocar o serviço e como o consumidor pode diferenciar serviços similares, oferecidos por diferentes fornecedores.

Em 2007, a W3C publicou (W3C, 2010) a especificação do WSDL 2.0 que agora convive ao lado do seu antecessor o WSDL 1.1 nos modelos de descritores de serviço de *Web Services* disponíveis no mercado. Isto acrescenta uma nova variável à variedade de opções para construção de um cliente para comunicação. A API (*Application Programming Interface*) para geração de *stubs* (conforme explicado no capítulo 3.3) ou o código dinâmico deve ser capaz de entender ambos os documentos descritores para conseguir efetuar a comunicação.

O programa que irá acessar o WSDL deve entender suas *tags* especiais do documento responsáveis por informar configurações e operações relativas ao serviço. Abaixo estão relacionadas as *tags* referentes a especificação 1.1 do WSDL conforme definição da W3C (2010), que continha mais elementos, alguns dos quais foram suprimidos na especificação 2.0, conforme a Figura 10 demonstra. São elas:

- **Types**: Informa os tipos de dados que estarão disponíveis dentro do documento. Pode-se criar elementos baseados em *XML Schema* dentro desta seção, ou importar schemas externos.
- **Message**: Define de forma abstrata a mensagem que irá trafegar, indicando dentro dos tipos de dados existentes, qual será utilizado.
- **PortType:** Responsável por informar quais operações estarão disponíveis no serviço. Esta *tag* utiliza-se da *tag message* para indicar quais são as mensagens de entrada, saída ou de erro que o método pode disparar.

- **Binding:** Define como o serviço será implementado, como por exemplo se utilizará SOAP e qual o estilo esperado.
- **Service**: Descreve o endereço para invocar o serviço. Esta *tag* é comumente editada pelo servidor de aplicação para poder dinamicamente alterar o endereço de serviço de acordo com a localização de sua implementação.

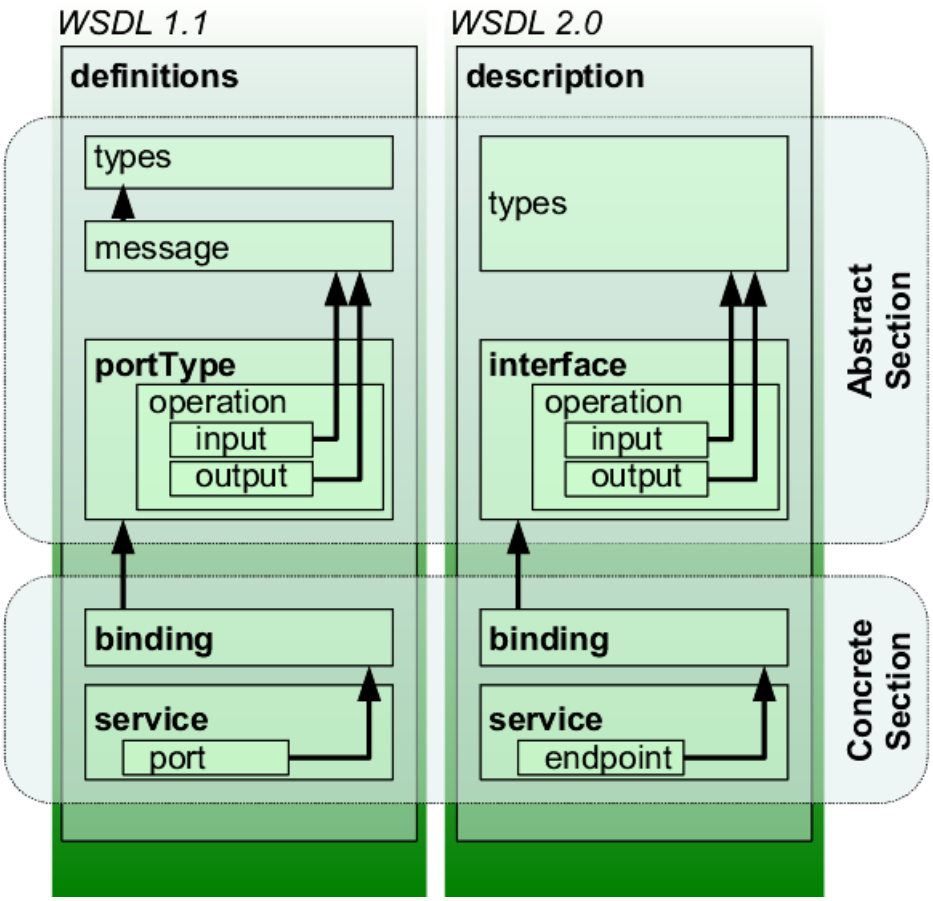


Figura 10 - Diferenças entre o WSDL 1.1 e 2.0

A Figura 10 demonstra algumas mudanças no modelo 2.0 do WSDL, como a remoção da marcação *message*, agora substituída por uma ligação direta ao elemento *types*. Além disto algumas nomenclaturas mudaram, agora dando maior sentido a funcionalidade, como as novas nomenclaturas "*description*"," *interface*" e "*endpoint*".

Além de definir a mensagem que deve trafegar, o WSDL também é responsável por informar em qual formato a mesma deve ser montada. BUTEK (2009), cita os 4 principais

modelos para composição de uma mensagem SOAP de acordo com o definido no WSDL. São eles:

- *RPC/encoded*;
- *RPC/literal*;
- *Document/literal*;
- *Document/encoded*.

Cada um destes modelos apresenta características distintas, conforme o próximo capítulo aborda.

3.1.4.1 RPC/encoded

Este modelo acrescenta nas *tags* XML que serão enviadas o tipo de dado que as mesmas estão representando dentro da *tag type* (W3C, 2010). As vantagens do mesmo são a semelhança da mensagem com o próprio documento WSDL e também por carregar o nome do método invocado na mensagem. As desvantagens são relativas a queda de performance justamente por os tipos de dados aparecerem na mensagem, causando assim uma sobrecarga na transmissão.

	WSDL
1	`<?xml version="1.0"?>`
2	`<message name="meuMetodoRequest">`
3	`    <part name="x" type="xsd:int"/>`
4	`    <part name="y" type="xsd:float"/>`
5	`</message>`
6	`<message name="empty"/>`
7	`<portType name="PT">`
8	`    <operation name="meuMetodo">`
9	`        <input message="meuMetodoRequest"/>`
10	`        <output message="empty"/>`
11	`    </operation>`
12	`</portType>`

Figura 11 - Representação de WSDL para RPC/encoded

	Mensagem SOAP
1	`<?xml version="1.0"?>`
2	`<soap:envelope>`
3	`  <soap:body>`
4	`  <meuMetodo>`
5	`  <x xsi:type="xsd:int">5</x>`
6	`  <y xsi:type="xsd:float">5.0</y>`
7	`  </meuMetodo>`
8	`  </soap:body>`
9	`</soap:envelope>`

Figura 12 - Mensagem SOAP RPC/encoded

3.1.4.2 RPC/ literal

Este modelo é bastante semelhante ao RPC/encoded, mas com a diferença da mensagem SOAP não conter as declarações de tipos de dados. Suas vantagens visíveis: diminuição da sobrecarga na rede, uma vez que uma quantidade menor de informações trafega, a existência da assinatura do método assim como o RPC/encoded facilitando assim a visualização de qual operação está sendo invocada e principalmente o fato deste modelo ser aprovado pela WS-I (*Web Service Interoperability Organization*) que é uma organização que define as melhores práticas para garantir a interoperabilidade para *Web Services* (MSDN, 2010). A desvantagem do modelo está relacionada com a dificuldade de validação da mensagem, uma vez que uma parte da mensagem (marcações SOAP) está definida no WSDL e o resto pode estar em um XML Schema externo. No código da Figura 13 é representado a mensagem SOAP. O WSDL para este modelo é igual ao RPC/encoded acima citado (diferenças somente na *tag binding*, não apresentada no exemplo, que conforme a introdução deste capítulo explica, informaria qual é o estilo da mensagem).

	Mensagem SOAP
1	`<?xml version="1.0"?>`
2	`<soap:envelope>`
3	`    <soap:body>`
4	`        <meuMetodo>`
5	`            <x>5</x>`
6	`            <y>5.0</y>`
7	`        </meuMetodo>`
8	`    </soap:body>`
9	`</soap:envelope>`

Figura 13 - Mensagem SOAP RPC/literal

3.1.4.3 Document/literal

Neste formato, a mensagem SOAP perde o nome do método e a forma de identificação de destino da mensagem passa a ser devido aos parâmetros informados. Consequentemente não pode haver dois métodos com os mesmos parâmetros. Alem disto, existe uma facilidade maior de validação da mensagem, pois o *XML Schema* esta anexado ao documento WSDL, viabilizando a validação da mensagem por inteiro (BUTEK, 2009).

Este modelo também é aprovado pela WS-I. As desvantagem são: complicação do documento WSDL e dificuldade de envio da mensagem uma vez que o nome do método não está mais incluído na mensagem. Nas Figuras 14 e 15 estão descritos o WSDL e a mensagem SOAP que representam este estilo de mensagem. Pode-se observar também a existência da *tag types* no WSDL que informa os tipos de dados da mensagem e o nome do elemento usado. Este elemento é referenciado na mensagem SOAP na linha 4 da Figura 15.

	WSDL
1	`<types>`
2	`      <schema>`
3	`            <element name="xElemento" type="xsd:int" />`
4	`      </schema>`
5	`</types>`
6	`<message name="myMethodRequest">`
7	`      <part name="x" element="xElemento" />`
8	`</message>`

```
 9  <message name="empty" />
10  <portType name="PT">
11        <operation name="meuMetodo">
12              <input message="meuMetodoRequest" />
13              <output message="empty" />
14        </operation>
15  </portType>
```

Figura 14 - Representação de WSDL para Document/literal

	Mensagem SOAP
1	`<?xml version="1.0"?>`
2	`<soap:envelope>`
3	`    <soap:body>`
4	`        <xElemento>5</xElemento>`
5	`    </soap:body>`
6	`</soap:envelope>`

Figura 15 - Mensagem SOAP Document/literal

3.1.4.4 Document/encoded

Neste modelo, os tipos de dados da mensagem vem junto de suas *tags* assim como o modelo RPC/Encoded e a assinatura do método deixa de existir semelhante ao modelo Document/literal (TYAGY, 2010) .

Segundo BUTEK (2009), este modelo não é utilizado pois não é aprovado pela WS-I. Uma prova desta afirmação é encontrada em ferramentas para criação de WSDL como o Eclipse, que não possibilitam a criação de um documento WSDL com este formato.

Independente do estilo escolhido para representação, um documento WSDL é a base para qualquer cliente de *Web Services* conseguir montar sua estrutura para comunicação. Clientes de *Web Services* podem ser ainda divididos em categorias, conforme o capítulo 3.3 irá abordar.

Para facilitar a interpretação de um documento WSDL dentro da programação, pode-se utilizar bibliotecas auxiliares como a EasyWSDL (http://easywsdl.ow2.org/), que abstraem o conteúdo do WSDL em objetos, conforme será descrito no capítulo a seguir.

3.1.5 EasyWSDL

Para que seja possível enviar uma mensagem para um *Web Service* é necessário previamente obter algumas informações do serviço tais como o endereço, qual PortType está usando (no caso de WSDL 1.1) e a assinatura do método. Estas informações são obtidas a partir do documento WSDL e suas estruturas auxiliares como namespaces e XML Schema

Para evitar ter que construir um parser XML que procure no WSDL por *tags* que descrevam as informações anteriormente citadas, procurou-se algumas alternativas de bibliotecas auxiliares para esta função. Dentre as encontradas estão: WSDL4J (http://sourceforge.net/projects/wsdl4j/), Apache Woden (http://ws.apache.org/woden/) e OW2 EasyWSDL.

Foram realizados testes com as implementações e as duas primeiras foram descartadas pelos seguintes motivos:

- WSDL4J: Suporte somente ao modelo de WSDL 1.1, dificultando em caso de ampliação do software. Outro motivo é que o projeto não recebe mais atualizações desde 2006;
- Apache Woden: Suporta ambos os modelos de WSDL, porem, trabalha convertendo o WSDL 1.1 para o modelo 2.0 e somente depois disto processa o documento. Como para a biblioteca isto é feito persistindo o arquivo convertido em disco, a performance é comprometida, principalmente levando em consideração a proposta do trabalho em ser um cliente dinâmico.

As características que eliminaram as outras bibliotecas não estão presentes na EasyWSDL. A mesma suporta ambos os modelos de WSDL, aplicando a sintaxe do modelo 2.0 em seu código (OW2, 2010). A leitura de um WSDL ocorre em memória, não importando qual sua versão e o projeto está em constante atualização. A mesma é distribuída pela OW2 Consortium que é uma comunidade open source para aplicativos de *middleware* (que fazem a mediação entre outros softwares).

Esta biblioteca está dividida em dois arquivos JAR (*Java Application Resource*), um responsável por trabalhar com o WSDL e a outro com o XML Schema, necessário para decodificar as informações relativas aos campos de tela e seus comportamentos.

Após a obtenção dos dados necessários para invocar um serviço, é necessário montar a mensagem. Para esta tarefa podemos utilizar diversos modelos conforme a seção a seguir aborda.

3.2 Modelos de cliente de Web Service

O acesso a um *Web Service* deve ser realizado por programas chamados de clientes ou *consumers* (consumidores). Estes programas são responsáveis pela comunicação, transmissão e interpretação dos dados que trafegam.

SAMPAIO (2006) classifica o desenvolvimento de um cliente de *Web Services* em três maneiras diferentes. São elas:

- **Stubs**: conjunto de artefatos gerados durante o desenvolvimento do Web Service. Estes artefatos, que para a linguagem Java são classes, convertem uma chamada de método da interface em uma requisição e uma resposta SOAP em valores de retorno para o cliente. A Figura 16 ilustra este modelo.

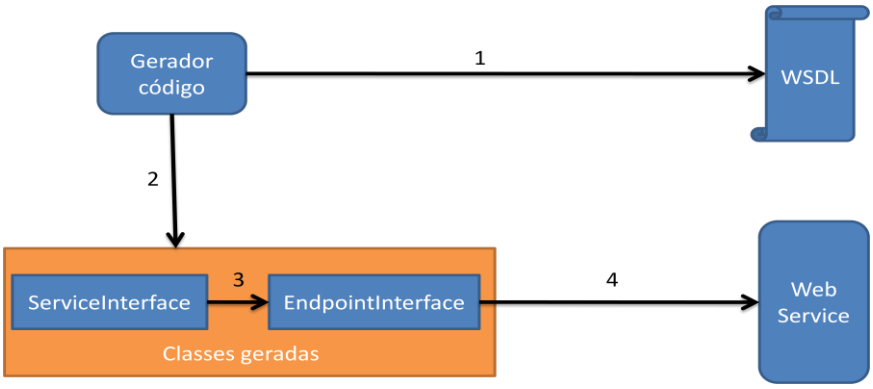


Figura 16 - Cliente Proxy

- **Dynamic Proxies**: O padrão de projeto Proxy define a criação de uma classe intermediária entre o cliente e o objeto destino para realizar uma conexão de forma eficiente (MERSKER, 2002). Esta classe intermediária contem a referência para o objeto real e pode acrescentar ou remover informações passadas pelo cliente para realizar a comunicação. No modelo de comunicação de *proxies* dinâmicos algumas classes (*Proxies*), são geradas dinamicamente baseadas no WSDL para acessar o serviço, conforme Figura 17. A principal diferença entre este modelo e Stubs é a adição da classe genérica para representação do serviço (Classe Service da Figura 17) . Esta classe após a primeira execução, armazena as informações do WSDL necessárias para realizar a comunicação, restando apenas o mapeamento para

a interface final do serviço (EndpointInterface), que representa os métodos disponíveis no serviço para que seja possível realizar uma conversão de tipos e transformar o objeto retornado em um objeto manipulável.

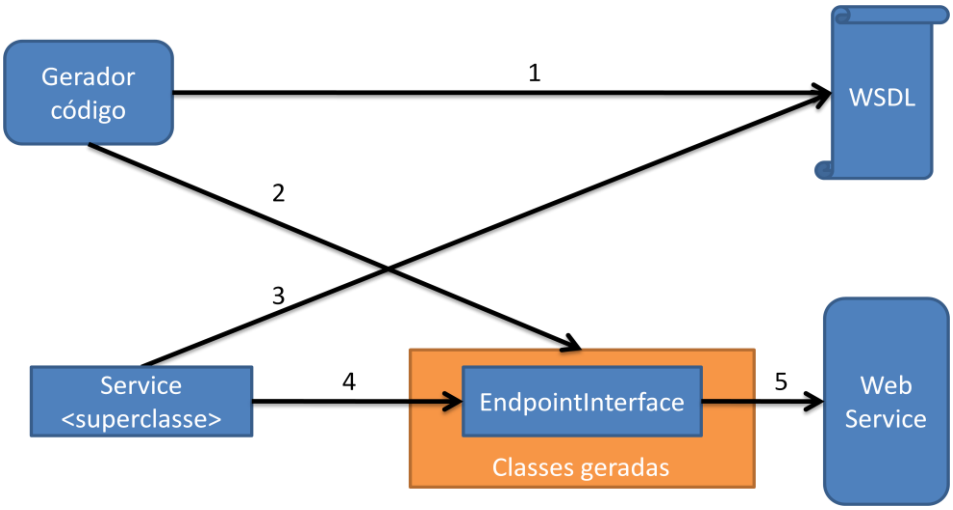


Figura 17 - Cliente Dynamic Proxy

- **Dynamic Invocation Interface**: Totalmente programático, permite ao desenvolvedor descobrir e invocar serviços em tempo de execução, atribuindo a montagem da requisição a classe que representa o serviço e que nos permite o envio da requisição, conforme Figura 18.

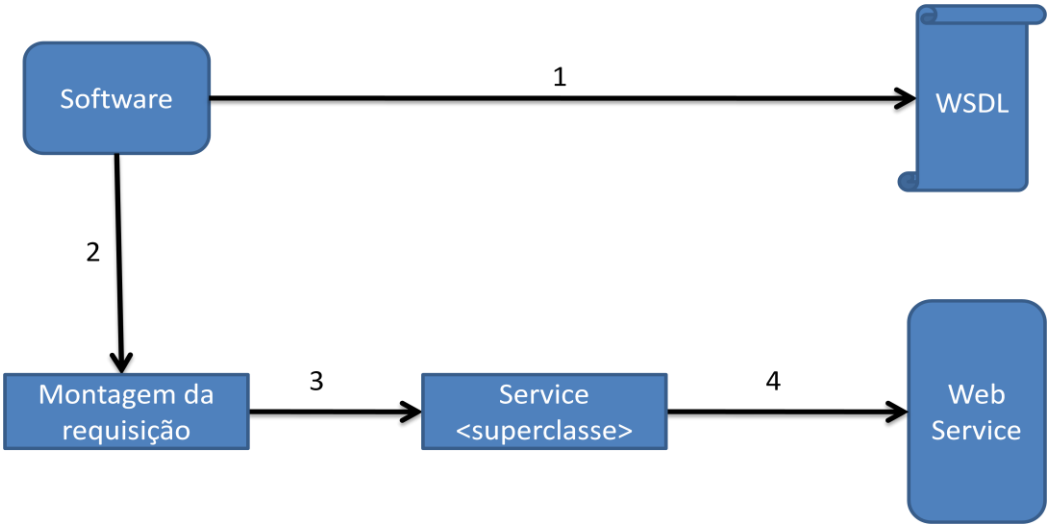


Figura 18 - Cliente Dinâmico

A ordem de performance dos tipos de implementação citados, segue a ordem acima citada. SAMPAIO (2006) explica cada um dos modelos citando a relação entre performance e portabilidade:

- Stubs é o modelo mais performático entre todos, uma vez que os artefatos são gerados com as informações completas do serviço, tendo assim, o conhecimento necessário sobre o que procurar no serviço destino, sem necessidade de analisar o WSDL antes de invocá-lo. Este tipo de modelo introduz o problema da portabilidade, uma vez que não temos como aplicar o mesmo programa a outro serviço.
- Proxies dinâmicos reduzem o acoplamento, mas sua performance é degradada por necessidade de gerar classes para acesso ao serviço em tempo de execução.
- Invocação de interface dinâmica exige um maior esforço de programação, uma vez que a chamada é feita completamente pelo desenvolvedor. Este modelo é o mais flexível de todos e sua performance esta diretamente relacionada com a forma de programação aplicada, utilização ou não de caches para otimizar a descoberta do serviço, etc. A vantagem deste modelo reside na possibilidade de construir um software capaz de se adaptar as necessidades do *Web Service*. Este modelo é onde este trabalho está fundamentado.

Tendo conhecimento do modelo de invocação de *Web Service* escolhido, procurou-se uma alternativa que atendesse o modelo e que facilitasse o desenvolvimento do mecanismo de envio de mensagens ao servidor. A alternativa encontrada, que é a utilização de JAX-WS *(Java API for XML Web Services*), pode ser visualizada na próxima seção.

3.3 JAX-WS

Pelo fato de *Web Services* funcionarem principalmente sobre a arquitetura HTTP, é possível que a invocação dos mesmos seja feito através da montagem de uma mensagem em formato texto e logo em seguida, realizando o envio através de um método POST do protocolo HTTP para o servidor destino. Este fluxo, apesar de viável, é bastante oneroso, uma

vez que é necessário que o desenvolvedor crie a mensagem trabalhando com Strings, criando as marcações XML manualmente, correndo o risco de não atender ao padrão especificado pelo XML e assim, invalidando a invocação do serviço.

Para facilitar este procedimento e não ter que trabalhar diretamente com Strings, utilizou-se neste trabalho a API JAX-WS, responsável por mapear requisições e respostas de serviços SOAP em objetos Java. Esta API, que é a implementação de referência distribuída pela SUN (JAXWS, 2010), possui uma série de funcionalidades, conforme descritas por HANSEN (2007). São elas:

- Invocação com interfaces Java;
- Invocação com XML;
- Binding SOAP;
- Binding HTTP;
- Conversão de exceções SOAP em exceções Java;
- Suporte aos estilos RPC/Literal e Document Literal do WSDL;
- Gerenciamento de threads no cliente;
- Invocação assíncrona;
- Operações de via única.

Características como Binding SOAP, suporte a RPC/Literal, Document/Literal e invocação com interfaces Java foram amplamente utilizados no presente trabalho, com o objetivo de suprimir a necessidade de construir o envelope SOAP para enviar a mensagem, ao mesmo tempo em que trabalha-se com uma sintaxe mais limpa dentro do código. Porém, o fator de maior relevância na escolha desta API em relação às outras disponíveis como AXIS (http://ws.apache.org/axis/) ou XFire (http://xfire.codehaus.org/) uma vez que ambas disponibilizam funcionalidades semelhantes, foi o fato desta API estar integrada na JVM nativamente nos pacotes javax.xml.ws e javax.xml.soap, dispensando assim, que fosse importado bibliotecas desnecessariamente para a aplicação.

Até o momento, foram apresentadas somente tecnologias envolvidas no acesso a um *Web Service*. Porém o presente trabalho também foca no desenvolvimento de uma interface gráfica que possibilita a usuário sem um grande conhecimento de *Web Services* consiga interagir com a tecnologia. Para tanto, optou-se pelo JAVAFX, que será apresentado detalhadamente na seção 3.4.

3.4 JAVAFX

O aumento da capacidade de processamento dos computadores pessoais propiciou o surgimento das RIA´s (*Rich Internet Applications*), que segundo CLARK (2009) são aplicações que permitem que uma boa parte do processamento da aplicação execute no lado cliente, inicialmente para aumentar a experiência do usuário, ou seja, aumentar o nível de interação que o usuário pode obter ao trabalhar com o sistema.

JavaFX, encontra-se na categoria de RIA, o que pode-se confirmar com a colocação de ANDERSON (2009) de que "JavaFX é uma tecnologia de software que permite criar aplicações RIA com mídia e conteúdo sobre uma grande quantidade de plataformas e dispositivos".

Esta tecnologia utiliza a JVM (*Java Virtual Machine*) como base para funcionamento, mas é necessário também a instalação de sua própria máquina virtual, que por ser dependente da JVM, tem seu tamanho significativamente menor do que a JVM.

A Figura 19, demonstra como é organizada a plataforma da tecnologia JavaFX e quais são os dispositivos suportados.

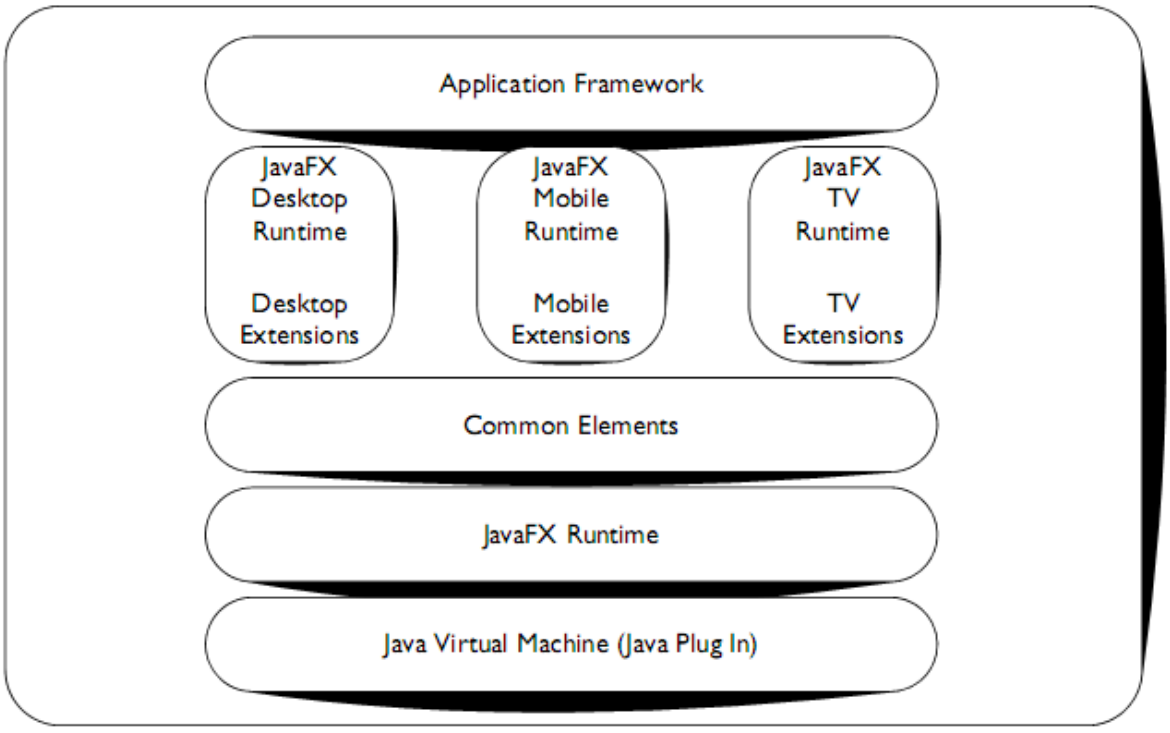


Figura 19 - Plataforma JavaFX, ANDERSON (2009)

A Figura 19 apresenta algumas características importantes da linguagem como a portabilidade, possibilitando que programas escritos utilizando esta tecnologia, funcionem localmente como aplicações *desktop*, via navegador ou até mesmo embarcadas em dispositivos como celulares e televisões.

A primeira versão do JavaFX foi publicada pela SUN Microsystens em dezembro de 2008 e, desde então, três outras versões foram lançadas, confirmando o grande investimento na linguagem devido ao curto período de tempo para atualização da mesma.

Esta linguagem tem a proposta de separar o trabalho de *design* gráfico da programação, partindo da afirmação que são duas habilidades diferentes e que raramente existem em um mesmo profissional.

CLARKE (2009), afirma que essa separação é possível no JavaFX graças a alguns elementos, tais como:

- Modelo de programação simplificado;
- Disponibilidade de componentes de interface prontos para uso e *frameworks* de suporte a criação de interfaces gráficas;
- Facilidade de atualização de aplicações gráficas existentes;
- Disposição de um ambiente multi-plataforma com a filosofia de "Escrever uma vez, execute em qualquer lugar".

Apesar do nome semelhante, JavaFX possui grandes diferenças da linguagem Java. Sua codificação é baseada na linguagem JavaFX Script, que é uma linguagem declarativa e fortemente tipada (DOEDERLEIN, 2009) . Porém as duas linguagens interagem de forma nativa e em qualquer momento de uma função ou declaração no JavaFX Script pode-se criar um objeto Java e relacioná-lo ao código. Esta característica garante a reusabilidade de programas escritos em Java e que tenham que ter suas interfaces alteradas, assim como supre as necessidades da linguagem JavaFX script enquanto esta amplia suas APIs.

Para demonstrar a diferença inicial entre as duas linguagens, foi criado um programa simples que exibe a mensagem "*Hello World*" utilizando JavaFX e recriado o mesmo utilizando a API Swing nativa da linguagem Java para construção de interfaces, conforme comparativo da Figura 20.

	JAVAFX	SWING
1	`package teste;`	`package teste;`
2	`import javafx.stage.Stage;`	`import java.awt.Dimension;`
3	`import javafx.scene.Scene;`	`import javax.swing.JFrame;`
4	`import javafx.scene.text.Text;`	`import javax.swing.JLabel;`
5		
6	`Stage {`	`public class HelloWorld {`
7	`title: "Hello World em JavaFX"`	`public static void main(String[] args) {`
8	`width: 300`	`JFrame frame =`
9	`height: 100`	`new JFrame("Hello World em Swing");`
10	`scene: Scene {`	`JLabel label = new JLabel("Hello World!");`
11	`content: Text {`	`Dimension d = new Dimension(300, 100);`
12	`x: 10, y: 30`	`frame.setSize(d);`
13	`content: "Hello World"`	`frame.add(label);`
14	`}`	`frame.setVisible(true);`
15	`}`	`}`
16	`}`	`}`

Figura 20 – Código comparativo entre JavaFX e Swing

Para o JavaFX, o objeto Stage representa o ponto de inicio do programa e este possui uma cena ou "*Scene*" a qual deve ser renderizada. Este trecho na linguagem Java é representada pelo método *main* e a declaração do JFrame como cena a ser renderizada.

Algumas alterações importantes foram realizadas também sobre como trabalhar com tipos de dados. Para JavaFX não existem tipos primitivos, os tipos de dados são derivados de *Object* ou seja, são objetos. Alguns destes valores, como o *Integer* são "tipos de valor", em que a igualdade de referência é equivalente a igualdade de valor (x == y somente se x.equals(y)). A Tabela 2, demonstra as equivalências de tipos de dados entre a linguagem Java e JavaFX.

Tabela 2 - Comparação entre os tipos de dados do JavaFX e Java. Fonte: DOEDERLEIN (2009)

Valores (value types)		
Comentários/Linguagem	JavaFX	Java
true ou false	Boolean	boolean
Inteiro, 8 bits sem sinal	Byte	byte
Inteiro, 16 bits com sinal	Short	short
Inteiro 32 bits com sinal	Integer	int
IEEE754, 32 bits	Number, Float	float
IEEE754, 64 bits	Double	double
String	String	String
Duração Temporal	Duration	N/A; costuma-se usar long

Também é possível no JavaFX Script a criação de funções e atribuí-las a variáveis e propriedades. No código da Figura 21 é demonstrado uma função JavaFX simples que retorna a soma de dois valores inteiros.

```
function soma(val1:Integer, val2: Integer){
      return val1 +val2;
}
```

Figura 21 - Exemplo de função JavaFX

No código da Figura 21 é possível verificar outra característica da linguagem, a inferência de tipo de dados, uma vez que não foi necessário declarar o tipo de retorno do método. Para variáveis também é possível, porém, após a primeira inferência a variável não pode ter seu tipo de dado alterado, pois conforme anteriormente citado, a linguagem é fortemente tipada.

Um exemplo de uma aplicação completa está na Figura 22. A mesma pode ser encontrada nos exemplos distribuídos pela SUN dentro da IDE Netbeans. A aplicação gira uma imagem para revelar uma segunda imagem atrás da primeira, conforme as Figuras 23 e 24 demonstram.

```
1  package fliptransition;
2
3  import javafx.animation.*;
4  import javafx.scene.input.*;
5  import javafx.scene.*;
6  import javafx.scene.effect.*;
7  import javafx.scene.shape.*;
8  import javafx.scene.paint.*;
9  import javafx.scene.image.*;
10 import javafx.stage.*;
11 import javafx.scene.text.*;
12 import java.lang.Math;
13 /*
14  * A Classe FlipView é um nodo customizado que mostra
15  * dois nodos filhos na frente e a trás , girando-os em uma transição 3D.
16  * Importante ressaltar que todo nodo customizado deve extender CustomNode
17  */
18 class FlipView extends CustomNode {
19
20  // Variáveis públicas que representam a parte da frente a parte de trás da imagem
21  public var frontNode:Node;
22  public var backNode:Node;
23
24  // Indica se a imagem está virada
25  var flipped = false;
26
27  // Animação de girar entre frente e trás, sendo atribuida a uma variável
28  var time = Math.PI/2;
29  public var anim = Timeline {
30    keyFrames: [
31        at(0s) { time=> Math.PI/2 tween Interpolator.LINEAR},
32        at(1s) { time=> -Math.PI/2 tween Interpolator.LINEAR},
33        KeyFrame {
34            time: 1.0s
35            action: function() { flipped = not flipped; }
36        }
37    ]
38  }
39
40  // Todo nodo customizado deve implementar o método create.
41  // Para sobrescrever um método, deve ser explicitamente
42  // declarado o modificador override.
43  // Neste método ao criar o nodo customizado FlipView, um grupo está sendo criado
44  // contendo os dois nodos que exibirão as imagens.
45  override public function create():Node {
```

```
46    return Group {
47     content: [
48      Group{content: backNode visible: bind (time<0) effect: bind getPT(time)},
49      Group{content: frontNode visible: bind (time>0) effect: bind getPT(time)},
50     ]
51    }
52  }
53   // Retorna a perspectiva de transformação.
54   // Calcula a mesma esticando as pontas da parte da frente
55   // e da parte de trás, de acordo com a curva do seno
56   // e do cosseno, multiplicado pelas constantes radius e back.
57   function getPT(t:Number):PerspectiveTransform {
58     var width = 200;
59     var height = 200;
60     var radius = width/2;
61     var back = height/10;
62     return PerspectiveTransform {
63         ulx: radius - Math.sin(t)*radius   uly: 0 - Math.cos(t)*back
64         urx: radius + Math.sin(t)*radius   ury: 0 + Math.cos(t)*back
65         lrx: radius + Math.sin(t)*radius   lry: height - Math.cos(t)*back
66         llx: radius - Math.sin(t)*radius   lly: height + Math.cos(t)*back
67     }
68   }
69  }
70
71  // Cria uma instancia da FlipView usando duas imagens localizadas
72  // no diretorio de execução do jar
73  var flip = FlipView {
74      translateX: 50
75      translateY: 40+50
76      backNode: ImageView { image: Image { url: "{__DIR__}lion1.png"  }  }
77      frontNode: ImageView { image: Image { url: "{__DIR__}lion2.png"  }  }
78  };
79
80  // Cria um objeto Stage, a qual armazena renderiza a cena a ser exibida.
81  Stage {
82    title: "Flip Transition"
83    scene: Scene {
84      fill: Color.BLACK
85      width: 300 height: 340
86      content: [
87        // A transição de girar
88        flip,
89        // O botão de clicar dentro de um agrupador
90        Group {
```

```
91          translateX: 50
92          translateY: 5
93          content:[
94            Rectangle { width: 200 height: 35
95                        fill: Color.rgb(100,100,100)
96                        stroke: Color.DARKGRAY },
97            Text { fill: Color.WHITE
98                   content: "Click Here to Flip"
99                   y: 25 x: 27 font: Font { size: 18 }},
100           Rectangle {
101                       width: 200 height: 35
102                       fill: Color.rgb(200,0,0,0.0)
103                        onMousePressed: function(e:MouseEvent) {
104                             if(flip.flipped) {
105                                 flip.anim.rate = -1.0;
106                                 flip.anim.time = 1s;
107                             } else {
108                                 flip.anim.rate = 1.0;
109                                 flip.anim.time = 0s;
110                             }
111                             flip.anim.play();
112                         }
113                     }
114                 ]
115             }
116         ]
117     }
118 }
```

Figura 22 - Aplicação completa em JavaFX

Ao clicar no retângulo com a mensagem “Click Here to Flip”, a imagem começa a girar, conforme a Figura 23 demonstra.

Figura 23 - Primeira imagem sendo girada

Após instantes, é exibido a imagem que estava atrás da Figura 23.

Figura 24 - Segunda imagem terminando o giro

Outras vantagens na utilização da tecnologia podem ser citados tais como a existência do recurso de *binding* para a associação entre propriedades entre si, *triggers* para disparar funções quando valores de variáveis forem alterados, entre outras. Porém, o simples fato da existência de integração nativa com a linguagem Java e também a grande freqüência de atualização da linguagem, tornaram atrativo o uso da mesma no presente trabalho.

Outro fator que resultou na escolha da linguagem JavaFX como linguagem de interface é a aposta na tecnologia RIA como o futuro do desenvolvimento de softwares das mais variadas naturezas tais como ERP (*Enterprise Resource Planning*), aplicativos para celular ou jogos, sejam eles locais ou disponibilizados na internet.

3.3.1 JFXtras

Conforme anteriormente citado, JavaFX é uma tecnologia bastante recente e em constante evolução. A mesma começou a atingir o nicho de mercado de aplicações comerciais a partir da liberação da versão 1.2.0, apresentada na JavaOne (Conferência de desenvolvedores) em 2009, quando apresentou componentes de uso comum como botões, *sliders*, *check box*, entre outros. Porem, até a versão 1.2.3 da linguagem, alguns componentes necessários para o desenvolvimento deste trabalho não foram liberados, tais como *combo-boxes*, tabelas, componentes refinados para leiaute e janelas de diálogo.

Para evitar ter que importar componentes da API Swing, bibliotecas paralelas como o JFXtras foram desenvolvidas por comunidades de forma a disponibilizar mais componentes para os desenvolvedores de aplicações em JavaFX.

No JFXtras todos os componentes foram desenvolvidos utilizando a própria linguagem JavaFX, associando componentes de *view* mais simples como retângulos, círculos a eventos disponíveis pela linguagem como *onClick* (ação de clicar), *onMouseOver* (ao passar o mouse por cima do objeto), entre outras. O código fonte do projeto está disponível para download juntamente com os binários no site da comunidade (www.jfxtras.org).

Dentre os componentes disponibilizados pela biblioteca, os mais utilizados neste trabalho foram os de leiaute XTable e MigLayout para posicionamento de campos em tela, o de controle XPicker (combo-box) e os de view RoundedRectangle e XDialog utilizados para arredondar janelas e abrir janelas popup.

Esta biblioteca está dividida em duas partes, uma responsável por trabalhar com o WSDL e a outra com o XML Schema, necessário para decodificar as informações relativas aos campos de tela e seus comportamentos.

4 PROJETO

Para atingir o objetivo deste trabalho, que é a construção de um cliente de *Web Service* dinâmico e gráfico, capaz de acessar uma grande variedade de serviços sem a necessidade de geração de código, conforme anteriormente citado, o trabalho estará fortemente baseado no documento descritor de *Web Services*, o arquivo WSDL. Para evitar a geração de artefatos, será utilizado o recurso de invocação dinâmica de interface como modelo de desenvolvimento.

Para descrever o software proposto, são utilizados nas sessões seguintes, alguns artefatos da metodologia ICONIX (ROSENBERG, 2005) para modelagem de softwares.

Conforme SILVA (2001), ICONIX é uma metodologia de desenvolvimento de software que é conduzida por casos de uso, utilizando-se da UML como linguagem de modelagem. A metodologia que é promovida pela empresa ICONIX *Software Engineering*, contempla as tarefas principais de análise de requisitos, análise e desenho preliminar, projeto e implementação.

4.1 Requisitos do projeto

Um software capaz de interagir através de uma interface gráfica com um *Web Service* apresenta alguns requisitos básicos para sua operação. Outros requisitos tornam-se interessantes para poder se trabalhar com as diversas relações 1:n que elementos de um WSDL mantêm entre si. Portanto, antes de dar início a construção do software, é necessário elencar os seus requisitos e criar seus casos de uso, que dentro da metodologia ICONIX são conceitos distintos, existindo uma relação de n:n entre si.

As Tabelas 3 e 4 demonstram respectivamente os requisitos funcionais e não funcionais de uma ferramenta desta natureza.

Tabela 3 - Requisitos funcionais

Código	Descrição
RF-001	O software deve ser capaz de estabelecer comunicação com um *Web Service*.
RF-002	O software deve permitir ao usuário a seleção de um serviço dentre os descritos no documento WSDL.
RF-003	O software deve permitir ao usuário a seleção da interface desejada para comunicação.
RF-004	O software deve permitir a seleção da operação desejada para processamento.
RF-005	O software deve permitir a entrada de dados de diversos tipos.
RF-006	O software deve poder enviar os dados para o *Web Service* destino.
RF-007	O software deve permitir a persistência dos dados de retorno da requisição.

Tabela 4 - Requisitos não-funcionais

Código	Descrição
RNF-001	Interface gráfica de fácil operação.
RNF-002	A aplicação deve rodar nos principais sistemas operacionais.

4.2 Casos de uso

Uma vez tendo os requisitos, é então criado um macro-diagrama de interação entre o usuário e os casos de uso do sistema. Vale ressaltar que um software desta natureza tem como ambiente de execução a internet, que depende de uma série de infra-estruturas, tais como roteadores, servidores e meios de comunicação.

Na Figura 25 estão relacionados os principais casos de uso em que o ator “Usuário”, único ator do sistema, está envolvido durante a comunicação com o serviço. Estes casos de uso refletem os requisitos funcionais apresentados na Tabela 3.

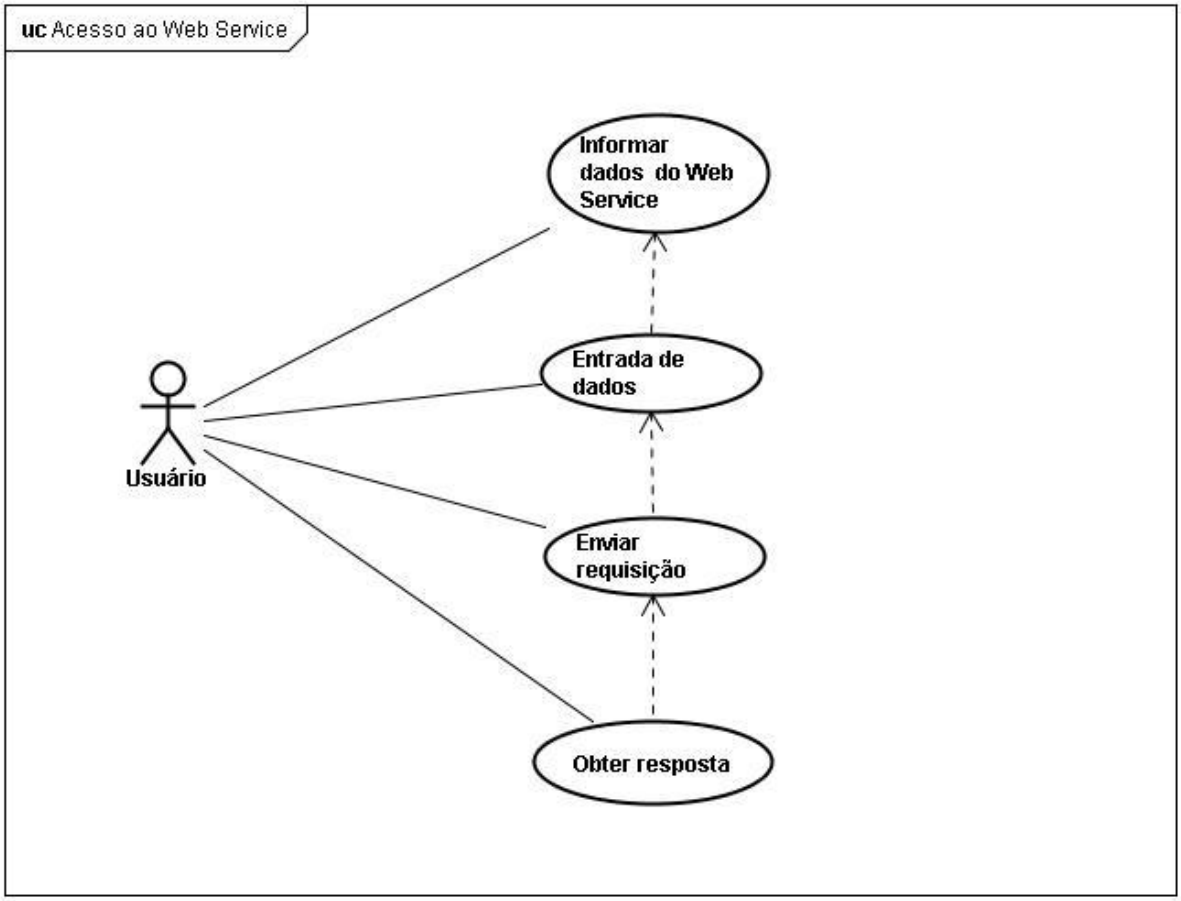
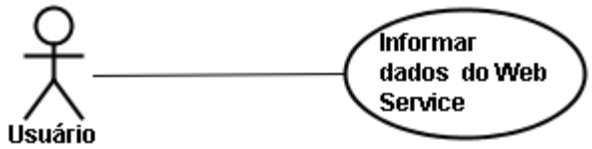


Figura 25 - Casos de uso que envolvem o ator usuário

A seguir são apresentados cada caso de uso apresentado na Figura 25.

CASO DE USO 001 – Informar dados do *Web Service* 	
Descrição	Prover informações para realizar a conexão com o *Web Service*.
Ator	Usuário
Pré-condições	Conhecimento do endereço do WSDL do *Web Service* desejado.
Fluxo Principal	1. Sistema apresenta uma tela para informar a URL do Serviço com a opção de enviar a solicitação. 2. Usuário preenche o campo da URL com o endereço do arquivo WSDL e clica em enviar.

	3. Sistema informa quais serviços estão disponíveis para a seleção do usuário. 4. Usuário seleciona um serviço dentre os listados. 5. Sistema informa quais interfaces estão disponíveis no serviço selecionado. 6. Usuário seleciona uma interface dentre as listadas. 7. Sistema informa as operações disponíveis na interface selecionada. 8. Usuário seleciona uma operação para iniciar a entrada de dados. 9. Sistema monta formulário relativo a operação selecionada.
Fluxo alternativo e exceções	**Item 2** 1. URL informada não corresponde a um endereço válido ou não existe o arquivo WSDL no endereço. 1.1. Sistema exibe mensagem avisando o usuário para reentrar com endereço do WSDL e mantêm o endereço anterior no campo. **Item 4** 1. Usuário troca a URL do serviço. 1.1. Sistema atualiza as opções de serviço, interface e método para as disponíveis no novo endereço. **Item 6** 1. Usuário troca a URL do serviço. 1.1. Sistema atualiza as opções de serviço, interface e método para as disponíveis no novo endereço. 2. Usuário troca o serviço selecionado. 2.1. Sistema atualiza as opções de interfaces e métodos disponíveis no novo serviço. **Item 8** 1. Usuário troca a URL do serviço 1.1. Sistema atualiza as opções de serviço, interface e método para as disponíveis no novo endereço. 2. Usuário troca o serviço selecionado 2.1. Sistema atualiza as opções de interfaces e métodos disponíveis no novo serviço. 3. Usuário troca a interface selecionada. 3.1. Sistema atualiza as opções de métodos disponíveis na nova interface.
Pós-condições	Tela com campos prontos para entrada de dados.

Figura 26 - Caso de uso "Informar dados do *Web Service*"

Cada caso de uso é dependente do caso anterior, uma vez que há necessidade de informações informadas em fluxos anteriores.

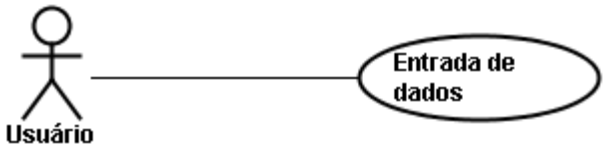
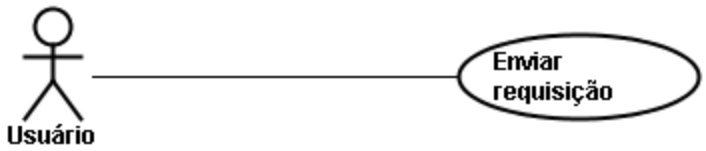
CASO DE USO 002 – Entrada de Dados	
	
Descrição	Preenchimento dos dados solicitados pelo Web Service destino
Ator	Usuário
Pré-condições	USE CASE 001
Fluxo Principal	1. Usuário preenche campos; 2. Sistema monta a mensagem no modelo esperado pelo Web Service;
Pós-condições	Campos disponíveis preenchidos

Figura 27 - Caso de uso "Entrada de dados"

Após o preenchimento dos campos conforme descrito na Figura 27, o usuário deve pressionar o botão responsável por enviar os dados ao servidor. Esta situação pode ser visualizada na Figura 28.

CASO DE USO 003 – Enviar requisição	
	
Descrição	Enviar requisição para o Web Service destino.
Ator	Usuário
Pré-condições	USE CASE 002
Fluxo Principal	1. Usuário solicita o envio da mensagem; 2. Sistema processa o envio; 3. Sistema apresenta a mensagem de retorno para o usuário, possibilitando sua

	persistência em disco;
Fluxo alternativo e exceções	**Item 2** 1. Ocorreram problemas de conexão com o Web Service. 1.1. Sistema apresenta a mensagem de erro e a opção de tentar reenvio;
Pós-condições	Mensagem de enviada para o Web Service.

Figura 28 - Caso de uso "Enviar requisição"

Após o envio da requisição, somente resta a interceptação do resultado obtido conforme o caso de uso da Figura 29 descreve.

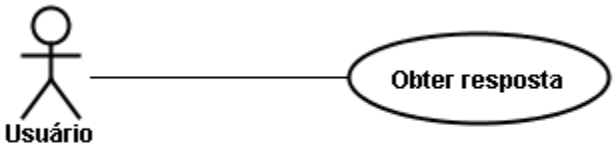
CASO DE USO 004 – Obter resposta	
	
Descrição	Receber e apresentar a mensagem de retorno do Web Service
Ator	Usuário
Pré-condições	USE CASE 003
Fluxo Principal	1. Usuário recebe a mensagem em tela; 2. Sistema disponibiliza persistência em disco da mensagem retornada;
Fluxo alternativo e exceções	**Item 1** 1. Ocorreram problemas de conexão com o Web Service. 1.1. Sistema apresenta a mensagem de erro;
Pós-condições	Mensagem de retorno apresentada para usuário em tela e pronta para persistência.

Figura 29 - Caso de uso "Obter resposta"

A existência de somente um ator envolvido nos casos de uso é pelo motivo da finalidade do software de realizar uma comunicação com um Web Service, independente de quem (ator humano), esteja tentando realizar esta comunicação.

4.3 Modelagem da solução

Os casos de uso apresentados demonstram os fluxos de interação do usuário com o sistema através de uma ação e uma resposta para cada situação.

Este conjunto de ações e respostas ocorre em cima de grandes etapas que o sistema deve realizar para atender a necessidade. As três grandes etapas do projeto são a interpretação do WSDL, a montagem da tela dinâmica e a montagem da requisição, conforme a seção a seguir descreve.

4.3.1 Interpretação do WSDL

Através de uma requisição HTTP, deve-se capturar o documento WSDL e desmontá-lo (este processo de desmontar, atribui-se o nome de parsear) em uma estrutura de dados possível de posterior leitura para montagem da tela. É importante nesta fase trabalhar com interfaces, para que seja possível trocar o mecanismo de compreensão do WSDL caso necessário. Esta situação não é difícil de ocorrer, uma vez que a especificação é atualizada freqüentemente e, por isso pode-se ter uma nova versão de WSDL ou até mesmo algum tipo de mecanismo proprietário no documento, que seja importante compreendê-lo (basicamente em necessidades comerciais do produto). Este fluxo pode ser melhor visualizado na Figura 30.

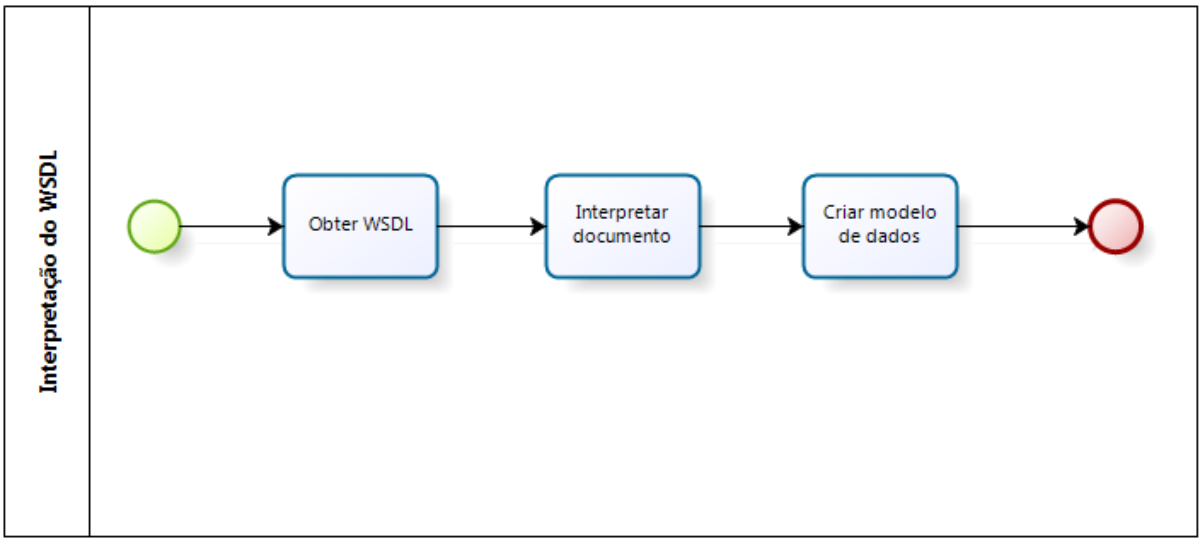


Figura 30 - Fluxo de interpretação do WSDL

4.3.2 Montagem da tela

Montar dinamicamente a tela baseada no modelo de dados anexado no WSDL é uma tarefa de mapeamento e testes. Conforme a Figura 7 mostrou, existe uma grande variedade de tipos de dados definidos para um XML Schema e para cada tipo de dado pode-se atribuir um componente de tela diferente. Outra situação possível de acontecer, é existir diversos itens encadeados como uma relação de nota fiscal e itens da nota. Os itens devem se tornar para a tela um componente que permita a adição de vários elementos de forma simples ao usuário e assim recursivamente para qualquer relação 1:N dentro dos itens. A interação completa dos elementos deste fluxo podem ser visualizados na Figura 31.

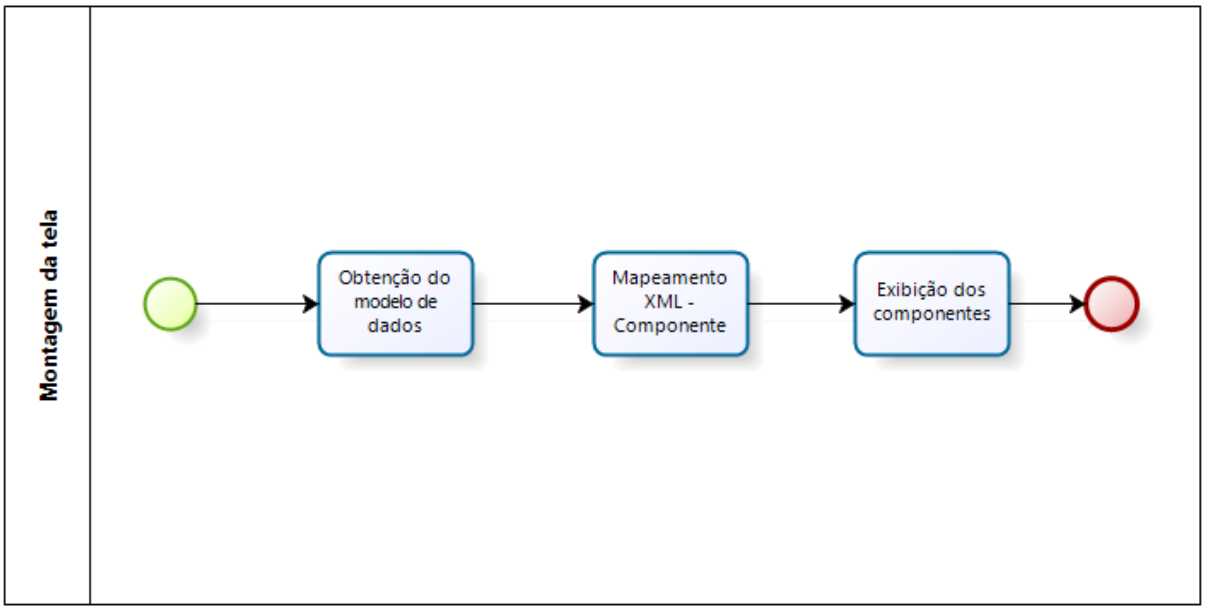


Figura 31 - Fluxo de montagem de tela

4.3.3 Montagem da requisição

Após ter os dados alimentados, deve-se realizar a operação reversa e atribuir os valores coletados a *tags* XML montando a requisição para envio. Dois problemas podem ser encontrados neste momento. O primeiro diz respeito a necessidade de saber os *namespaces* de cada *tag*. Dentro de uma mensagem XML pode existir mais de um *namespace* ou seja, pode existir uma *tag* que pertença ao *namespace* do módulo de compras e outra que pertença ao módulo de vendas. Na hora de montar uma mensagem, é necessário ter esta informação. Outra dificuldade está relacionada sobre os modelos de requisição SOAP que podem ser montados

dentre os 4 existentes, conforme a seção 3.1.4 apresentou. No presente trabalho, o foco somente foi aplicado nos dois modelos mais utilizados, que é o Document/literal e o RPC/literal.

A Figura 32 demonstra como ocorre o processo de montagem de uma requisição.

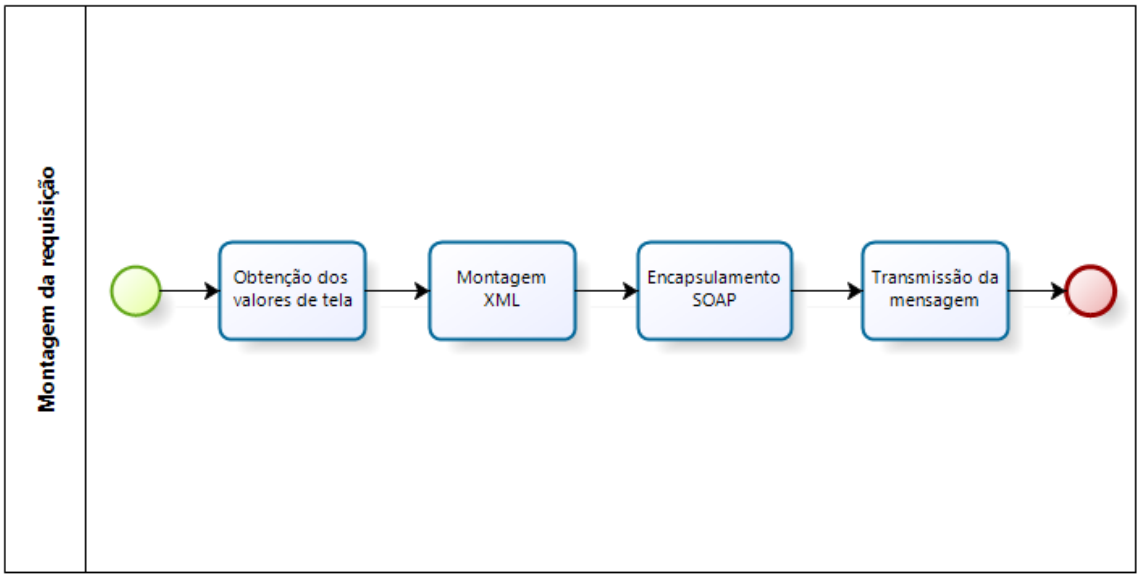


Figura 32 - Fluxo de montagem de requisição

Para cada um dos itens da Figura 32 apresentados, existem diversas formas de implementação, algumas com auxílio de bibliotecas e outras não. Como exemplo pode-se citar para o fluxo de interpretação do WSDL, o uso de bibliotecas como a Apache Woden ou OW2 EasyWSDL, atribuindo os dados resultantes destas API´s em modelos de dados que facilitem o uso dentro do software.

A junção de todos os itens é o último problema que o software deve realizar. Trocar a implementação de um dos itens não deve impactar nos outros itens. Para tanto elaborou-se um modelo de domínio que atenda a premissa de separar as responsabilidades entre as funcionalidades esperadas, conforme pode ser visto na próxima seção.

4.4 Modelo de domínio

Conforme anteriormente explicado o seguinte modelo de domínio preza a separação das responsabilidades dos itens “Interpretação do WSDL”, “Montagem da tela” e “Montagem da requisição”.

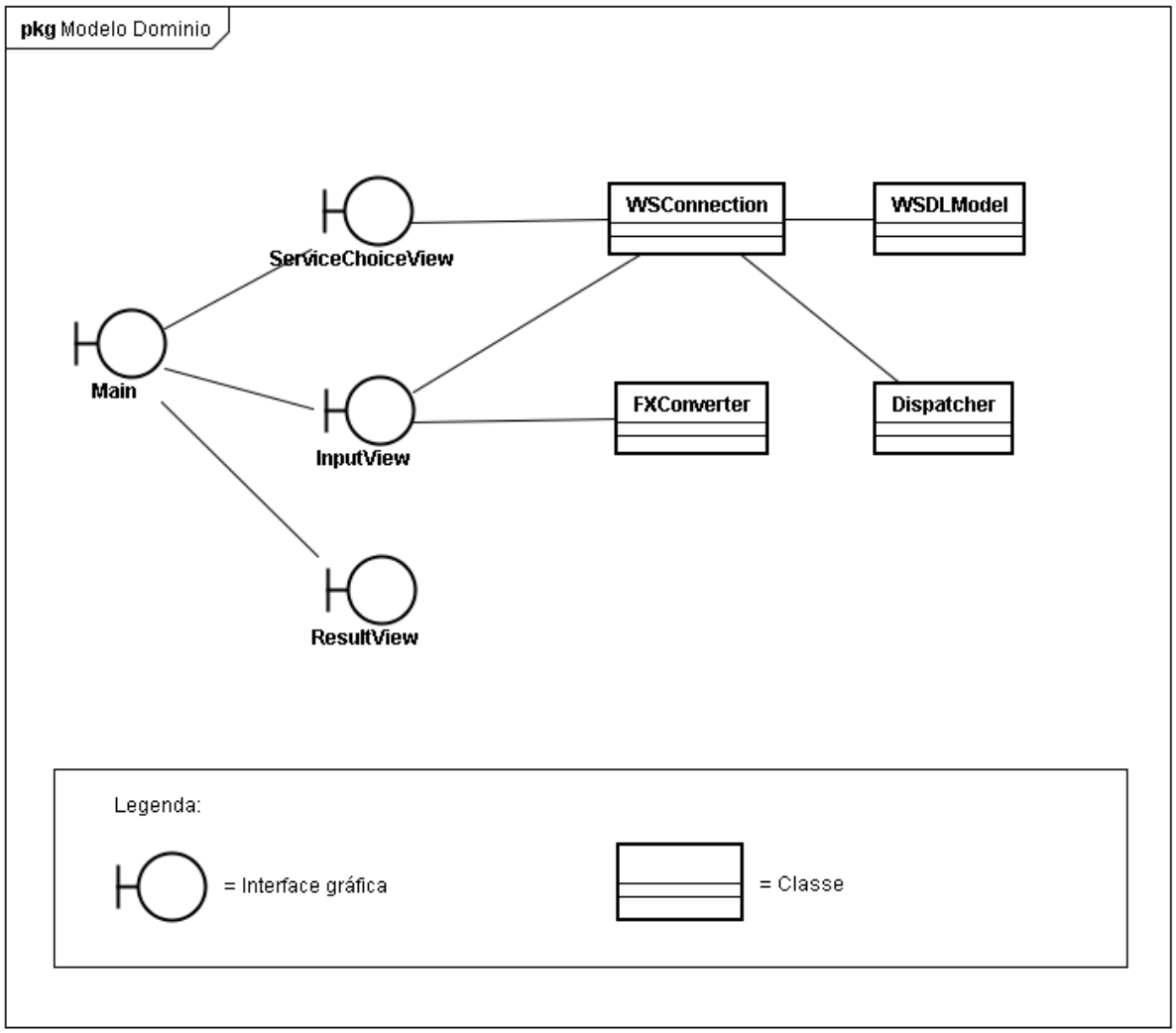


Figura 33 - Modelo de domínio do software

Cada elemento deste modelo de domínio exerce funções específicas. A seguir são relacionados cada um destes elementos.

- **Main:** Container para todas as *views* que serão desenvolvidas.

- **ServiceChoiceView**: Esta interface é responsável por apresentar as opções de interfaces, serviços e métodos disponíveis no WSDL destino informado pelo usuário.
- **InputView**: Interface responsável por apresentar os campos para preenchimento do usuário. A mesma irá tratar consistências básicas apresentando informações de validação.
- **ResultView**: Interface responsável por exibir a resposta do servidor após o envio da mensagem.
- **WSConnection**: Classe que irá realizar a comunicação com o *Web Service* e também obter o documento WSDL, fornecendo a classe WSDLModel que representa o modelo de dados do documento WSDL e a classe Dispacher para envio dos dados.
- **WSDLModel**: Representação do documento WSDL em uma estrutura que viabilize sua exibição na interface ServiceChoiceView.
- **Dispacher**: Classe responsável por enviar os dados ao servidor e obter o retorno. No projeto ela representa uma interface para que seja possível ter diversas implementações.
- **FXConverter:** Classe responsável por criar os campos dinamicamente para serem exibidos para o usuário na interface InputView. A mesma utilizará o modelo de dados disponibilizados pela classe WSConnection.

Tendo conhecido a lógica básica dos artefatos envolvidos no software, pode-se montar um panorama da arquitetura esperada deste sistema, conforme a seção a seguir aborda.

4.5 Arquitetura

O software proposto é baseado no modelo cliente-servidor, no qual o software localizado em uma máquina qualquer, dispara uma requisição para um servidor remoto, aguardando uma resposta (processo síncrono). O conjunto de classes anteriormente apresentado constitui o que o diagrama de arquitetura representado na Figura 34 demonstra.

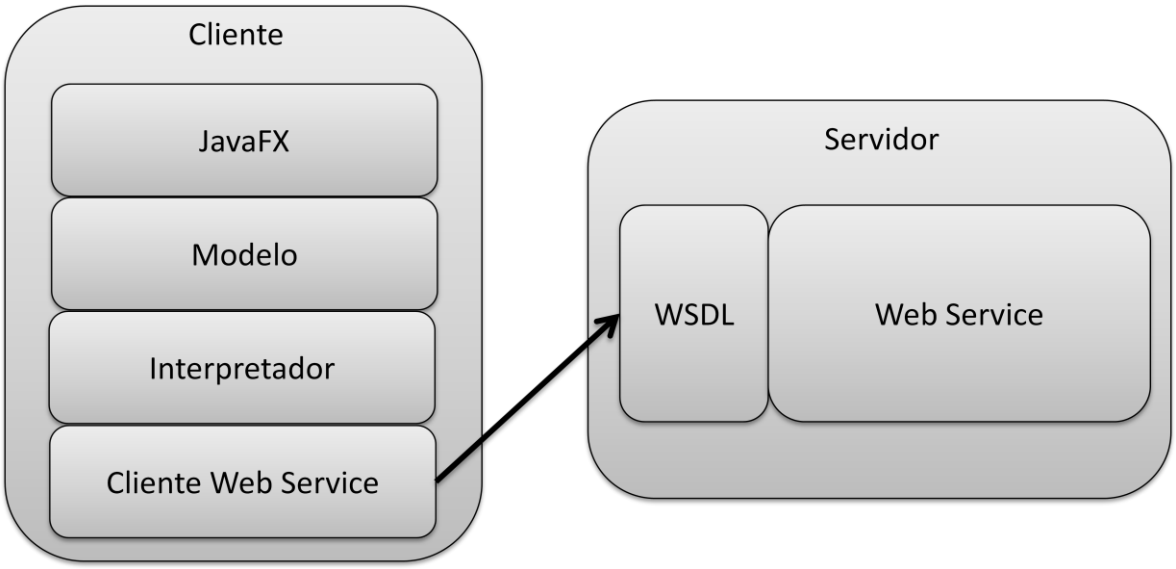


Figura 34 - Diagrama de arquitetura

O objetivo é separar as responsabilidades de cada elemento da aplicação, facilitando o objetivo final, que é possibilitar que o usuário conecte-se dinamicamente ao *Web Service* e envie uma mensagem após informar os valores esperados pelo serviço em uma interface gráfica. Pode-se criar uma associação dos componente da arquitetura (do lado cliente) e o modelo de domínio, conforme a Tabela 5 demonstra

Tabela 5 - Relação componente de arquitetura x classes de domínio

Componente de arquitetura	Classes de domínio
JavaFX	MainView, ServiceChoiceView, InputView, ResultView e FXConverter
Modelo	WSConnection e WSDLModel
Interpretador	WSConnection
Cliente de *Web Service*	WSConnection e Dispatcher

Com esta relação de componente-classe, pode-se verificar a grande utilização da classe WSConnection. Isto deve-se ao fato da existência de estruturas auxiliares que são criadas a partir da mesma para compartilhamento com outras classes, a exemplo da classe WSDLModel, que é criada após primeira conexão com o servidor onde são obtidas as

informações relativas a forma com a qual uma mensagem deve ser enviada para que o servidor a aceite como válida.

Para alcançar o objetivo da arquitetura demonstrada, procurou-se a utilização de boas práticas de programação conhecido como padrões de projeto.

4.5.1 Design Patterns

O desenvolvimento do presente trabalho, levou em consideração os requisitos de performance e reutilização para garantir a sobrevida do sistema e evitar problemas na migração de tecnologia, caso fosse necessário. Desde o desenvolvimento da interface, que evitou a utilização de recursos especiais da linguagem JavaFX para manipulação de arquivos e *Web Services*, deixando esta funcionalidade para a camada de negócios desenvolvida em Java, até os cuidados de se trabalhar no maior nível de abstração possível na interpretação e comunicação com *Web Services*.

Para garantir os requisitos apresentados, procurou-se aplicar padrões de projeto já consagrados e amplamente documentados, também são chamados de *Design Patterns (Padrões de Projeto)*.

Os três padrões de projeto que foram explicitamente utilizados neste trabalho são o Singleton, *Builder* e MVC (*Model View Controller*), apresentados por GAMMA(2000):

- *Singleton*: O padrão singleton é utilizado para garantir que uma classe tenha somente uma instância e que somente tenha um ponto de acesso. Neste projeto este padrão está visível nas classes GlobalFXVariables.fx e na GlobalBinds.fx, ambas classes JavaFX, utilizadas para manter informações que são úteis para o restante da aplicação.

- *Builder*: O padrão *Builder* tem como objetivo separar a construção de um objeto complexo de sua representação, de modo que possa se usar o mesmo mecanismo para criar diversas representações. No presente trabalho, tanto a criação do objeto responsável por processar o WSDL, como a criação do objeto responsável por enviar a mensagem SOAP ao servidor, foram realizadas utilizando este padrão devido

a quantidade de alternativas e bibliotecas existentes para realizar as tarefas (conforme explicado no cap 3.1.5 e 3.3) .

- MVC: Este padrão é composto por três tipos de objetos. O modelo (*Model*) que representa a regra de negócio e os dados da aplicação, a visão (*View*) que é a interface com usuário e o controlador (*Controller*) que define como a interface reage as entradas de informações por parte do usuário. A utilização deste padrão surgiu como uma necessidade para gerenciamento da invocação de telas. O padrão aplicado a JavaFX é baseado no artigo de DOEDERLEIN (2009), em que o mesmo construiu uma aplicação simples de cadastramento de produtos e exibição de gráficos utilizando MVC, demonstrando a possibilidade de JavaFX ser utilizado em outros ambientes além da exibição de multimídia. Neste trabalho o MVC está presente em relação a todas as telas do sistema.

Após a apresentação da arquitetura esperada, pode-se elaborar o diagrama de pacotes dos artefatos envolvidos no software, conforme a seção 4.6 demonstra.

4.6 Diagrama de pacotes

O projeto desenvolvido tem como premissa a separação da lógica envolvida para comunicação e interpretação de serviços da tecnologia utilizada para *view*.

Para tanto o a estrutura de pacotes da aplicação segue a estrutura definida na Figura 35.

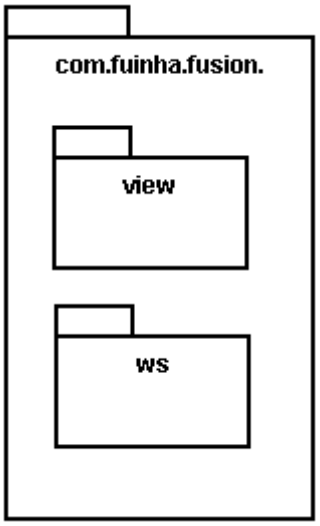


Figura 35 - Primeiro nível da arquitetura de pacotes

O objetivo nesta estrutura é que os códigos fontes da aplicação fiquem abaixo da estrutura padrão "com.fuinha.fusion" que identifica o produto desenvolvido. No nível abaixo deste pacote padrão temos o pacote view onde estão contidas todas as estruturas utilizadas dentro da camada de visualização, sejam elas classes JavaFX ou até mesmo estruturas Java auxiliares, utilizadas como suporte para classes JavaFX. No pacote ws é onde toda a lógica de processamento de requisição e interpretação de documentos WSDL encontra-se. O código fonte encontrado abaixo deste nível são somente classes Java, uma vez que JavaFX deve ser somente utilizado como estrutura da camada de apresentação.

Dentro da camada *view* uma série de pacotes foram criados de forma a criar uma separação dos elementos que são apresentados em tela dos componentes reutilizáveis conforme a Figura 36 demonstra.

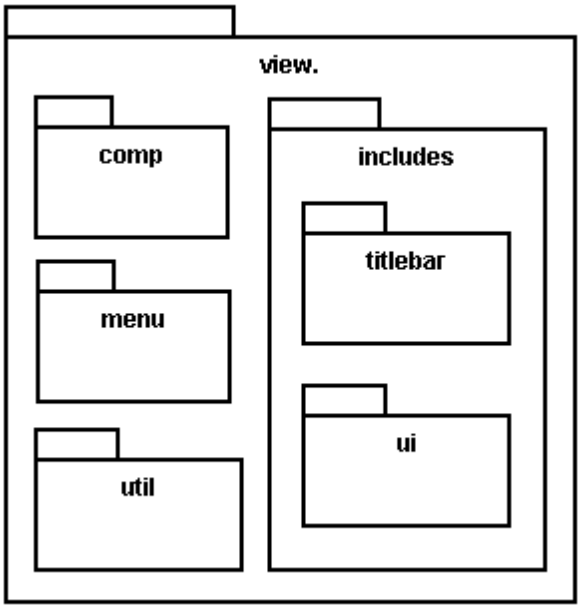


Figura 36 - Pacotes da camada view

Dentro deste nível encontramos pacotes com responsabilidades distintas tais como:

- comp: Pacote responsável por armazenar componentes reutilizáveis da aplicação. Todos os componente aqui contidos foram desenvolvidos exclusivamente para a aplicação e encapsulam outros componentes tanto nativos do JavaFX como componentes de bibliotecas como JFXTras.
- includes: Pacote destinado a implementações que estão anexadas ao layout principal da aplicação.
- titlebar: Neste pacote está todos os elementos utilizados na montagem da barra de título da aplicação, assim como os elementos que possibilitam que a janela da aplicação seja arrastada ao clicar na barra de título.
- menu: Assim como para o pacote titlebar, este pacote contem elementos necessários para exibir o menu e sua animação nos ícones.
- ui: Pacotes responsável por exibir as telas finais de apresentação de informações ao usuário e todas as classes que contribuem para a apresentação das mesmas.
- util : Pacote de classes utilitárias para a camada de apresentação.

Para a camada de interação *Web Services*, os pacotes se dividem de forma a separar as diversas funcionalidades envolvidas no processo de interpretação de um serviço até o envio da mensagem desejada, conforme a Figura 37 demonstra.

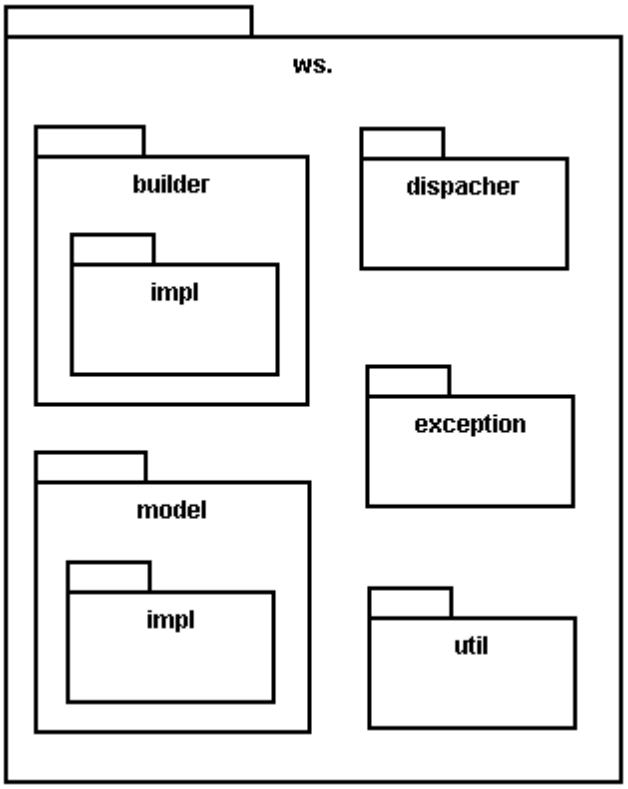


Figura 37 - Pacotes da camada de lógica

Os objetivos dos pacotes apresentados na cama de lógica são:

- builder : Neste pacote encontram-se todas as classes responsáveis pela construção de objetos complexos, que implementam o *Design Pattern Builder*. Na raiz deste pacote encontram-se as classes abstratas utilizadas pelo *Design Pattern Builder*.
- impl : Neste subpacote encontram-se as classes de implementação do método build (contruir) de cada uma das classes abstratas criadas no nível de pacotes acima.
- model : Pacote aonde todos os modelos de dados que trafegam pela aplicação estão contidos. A interação da camada de interface ocorre em cima dos modelos de dados existentes neste pacote. Na raiz deste pacote encontra-se a

interface WSDLModel responsável por responder as solicitações de tela de seleção do serviço desejado.

- impl : Subpacote aonde encontram-se todas as classes utilizadas pela interface WSDLModel.
- dispacher : Pacote de classes responsáveis por despachar a mensagem interpretada do modelo de dados ao servidor destino .
- exception : Pacote onde se encontram as exceções personalizadas tratadas pelo sistema.
- util : Pacote de classes utilitárias da camada de lógica da aplicação.

4.7 Diagrama de classes

As próximas sessões explicarão os pontos chaves do software desenvolvido através de diagramas de classes, iniciando pela diagramação dos componentes desenvolvidos para apresentação de informações ao usuário. Vale lembrar que um código JavaFX pode ser escrito em formato de classe ou script. Apesar de poucos os casos em que foi utilizado somente script no projeto, os diagramas de classes a serem apresentados tratarão os mesmos como classes.

4.7.1 Componentes de entrada de dados

Diversos componentes foram desenvolvidos para ser utilizados no projeto, sendo os principais voltados a entrada de dados, conforme a Figura 38 demonstra.

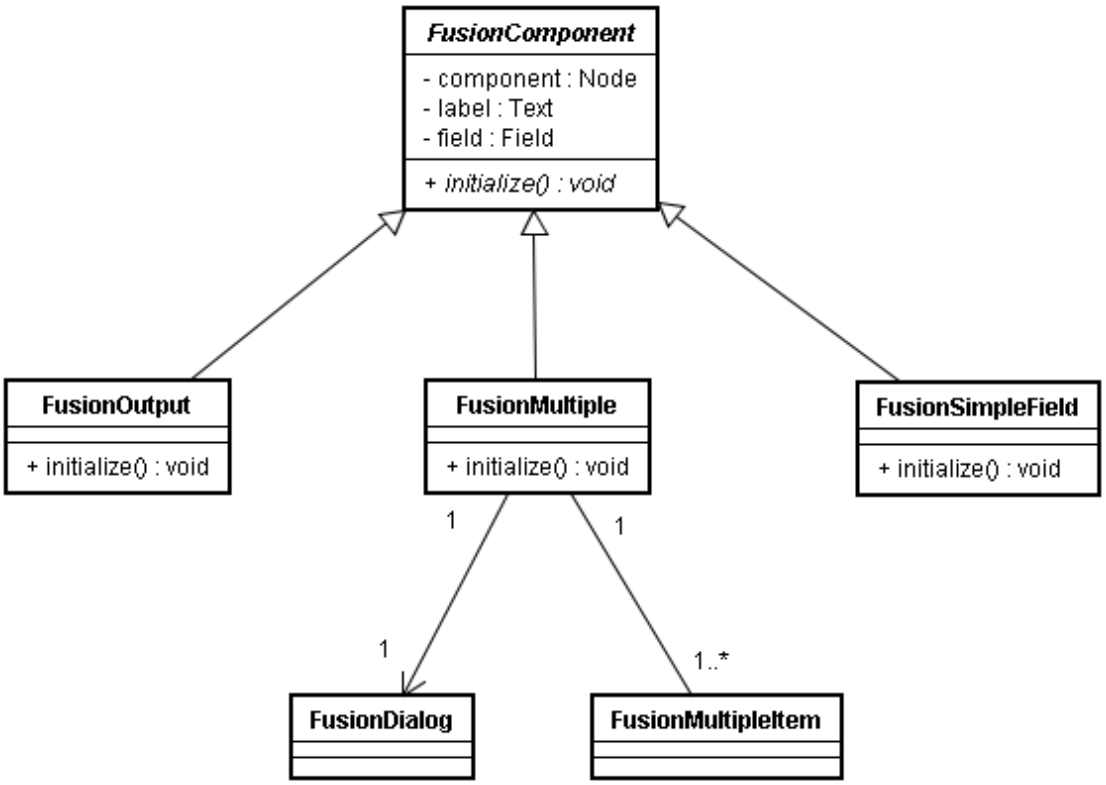


Figura 38 - Diagrama de classe de componentes de tela

Os componentes de entrada de dados herdam da classe FusionComponent para reaproveitar funcionalidades comuns a todos e para obrigar a implementação do método abstrato initialize(), que é utilizado para criar o componente. O JavaFX possui uma estrutura de criação de nodos customizados, que não foi utilizado no projeto pois o mesmo somente retorna um nodo, sendo assim inadequado a exibição em uma estrutura de *grid* em duas colunas como é o layout da tela de entrada de dados do projeto FUSION.

A classe abstrata FusionComponent possui os seguintes atributos:

- component: Atributo do tipo Node (javafx.scene.Node) a qual representa qualquer componente de interface gráfica, simples como um botão ou até mesmo, componentes customizados, que podem agregar vários componentes.
- label: *Label* (identificador) do componente. Ao invés de criar um componente do tipo *Label* em todas as telas, utilizou-se um atributo para desempenhar este papel.
- field: Campo recebido do modelo de dados gerado após a interpretação do WSDL. Através deste atributo é criado os atributos *label* e *component*.

As implementações da classe FusionComponent são:

- FusionOutput: Componente para exibir somente o *label*, com o seu atributo component nulo, deixando assim um separador para quando a *tag* XML esperada é somente um agrupador.
- FusionSimpleField: Componente de campo texto simples.
- FusionMultiple: Componente para exibição de listagens, com opção de adicionar , remover e editar a linha corrente abrindo uma Dialog.

Também há outros componentes, desenvolvidos como auxiliares para as diversas situações encontradas no desenvolvimento de uma interface gráfica, que fazem parte do projeto. A seção 4.7.2 aborda esses componentes.

4.7.2 Componentes auxiliares

Alguns componentes foram criados para melhorar a interação com o usuário ao mesmo tempo em que acrescentaram funcionalidades inexistentes nativamente no JavaFX 1.2. O diagrama da Figura 39 apresenta estes componentes.

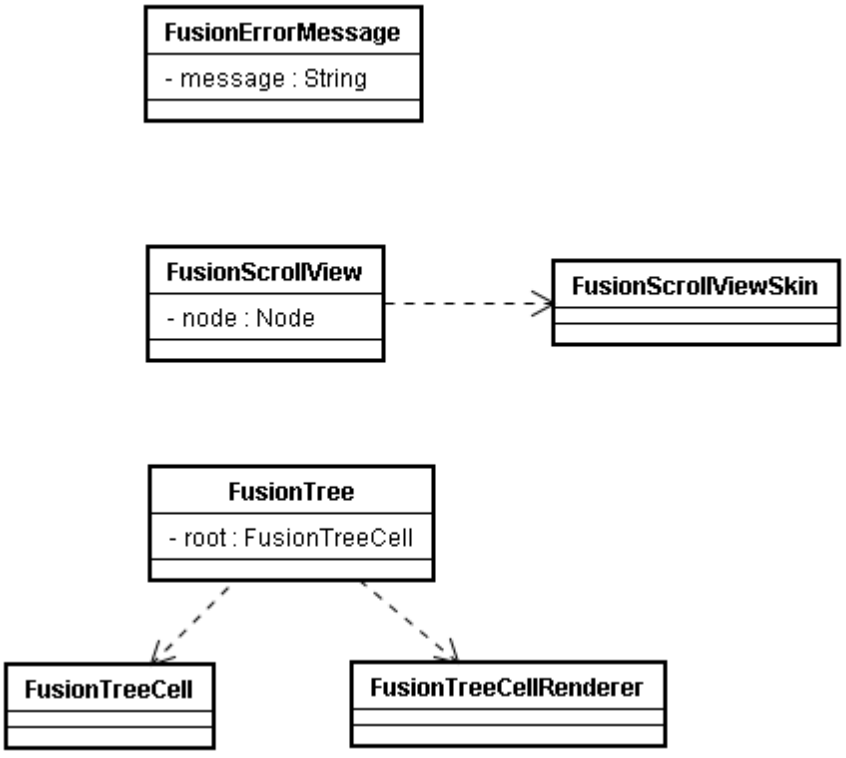


Figura 39 - Componentes auxiliares

A primeira necessidade não disponível dentro da API do JavaFX 1.2 é uma barra de rolagem lateral para quando a quantidade de campos necessários a serem informados excedesse o tamanho de tela. Para resolver este primeiro problema, adaptou-se da classe XScrollView do projeto JFXtras o componente FusionScrollView. A diferença da classe adaptada é que a mesma consegue deixar seu plano de fundo transparente, adaptando-se melhor a qualquer cor de fundo que seja aplicada no projeto. Esta alteração foi realizada principalmente na classe FusionScrollViewSkin, que representa as características gráficas do componente.

A segunda necessidade encontrada para o projeto foi a de renderizar uma árvore para exibição do XML recebido do servidor. Assim como o requisito anterior, até a versão 1.2 do JavaFX não existia componente de árvore nativo, portanto era necessário invocar a API Swing e utilizar o objeto JTree. Este procedimento foi realizado na classe FusionTree, FusionTreeCell e FusionTreeCellRenderer. A FusionTree exige que seja retornado um nodo raiz para que a árvore seja montado. Este nodo raiz e seus filhos são do tipo FusionTreeCell. A classe FusionTreeCellRenderer foi utilizada para deixar o plano de fundo transparente, assim como foi realizado no componente anterior.

A ultima necessidade é a de exibir mensagens de erro disparadas pelo sistema. O JavaFX 1.2 possui um componente para exibição de mensagens em uma janela de diálogo, porém, para que a janela da mensagem se assemelhasse as outras janelas de dialogo abertas pelo sistema, um componente customizado foi criado, com o nome de FusionErrorMessage.

4.7.3 Telas do sistema

Todas as telas desenvolvidas para o projeto FUSION, herdam da superclasse FusionView, conforme mostra a Figura 39.

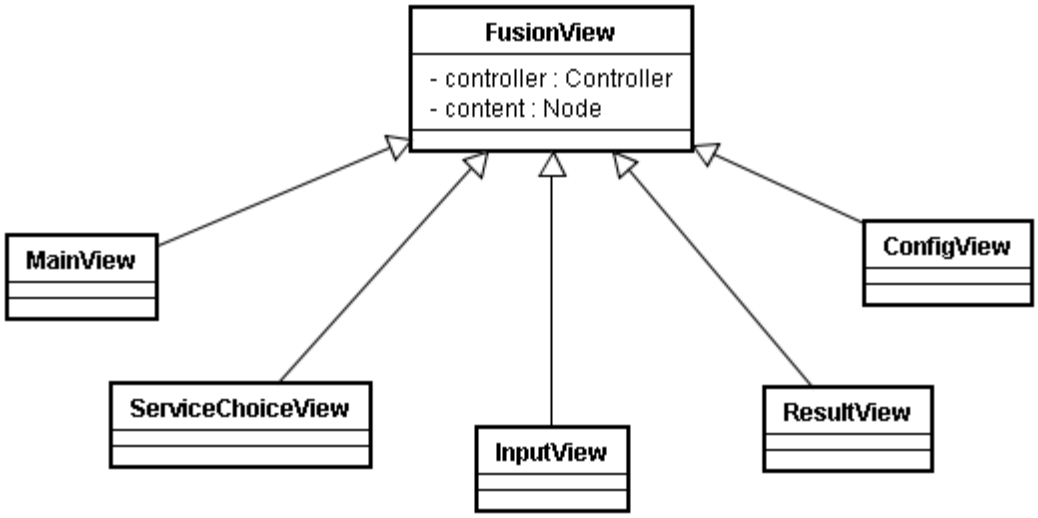


Figura 40 - Telas do Sistema

Os dois atributos da FusionView são importantes para todas as telas. São eles:

- content: Representa o conteúdo sendo exibido no momento, toda a tela que extender FusionView, deve ao seu final sobrescrever essa variável com o conteúdo que deseja exibir, utilizando a sintaxe: override def content = bind <NOVO_NODO>.
- controller: Variável que representa a classe Controller do modelo MVC, responsável pela invocação de qualquer outra tela do sistema. Para tanto é utilizado a sintaxe: controller.show<NOME_VIEW>(<PARAMETRO>).

O real conteúdo de cada tela é construído dentro da cada uma e atribuído a variável content, que quando substituída atualiza a visualização.

Além das *views* já apresentadas no modelo de domínio, neste diagrama encontramos outras views, desenvolvidas para completar as funcionalidades do software. São elas:

- MainView: Tela desenvolvida para exibir os dados iniciais do programa, uma vez que é a primeira tela a ser invocada. Inicialmente estará informando dados como autor do projeto e a universidade em que o projeto foi realizado.
- ConfigView: Tela responsável pelas configurações do software. Inicialmente a única configuração disponível é a o controle de cor da aplicação, mas futuramente, outras funcionalidades são esperadas que necessitarão desta view para funcionamento.

A partir da tela de escolha do serviço (ServiceChoiceView) o sistema passa a interagir com um modelo de dados construído com o objetivo de trafegar em todas as camadas da aplicação, podendo ter seu conteúdo alterado a qualquer momento, conforme a necessidade. Este modelo de dados está descrito na seção a seguir

4.7.4 Modelo de dados

O conjunto de classes que será descrito a seguir tem como objetivo abstrair a complexidade do documento WSDL e do XML Schema em elementos que sejam fundamentais para o funcionamento da aplicação, criando uma estrutura que viabilize a dinamicidade da mesma. Este conjunto de classes constitui o modelo de dados da aplicação, conforme a Figura 41 demonstra.

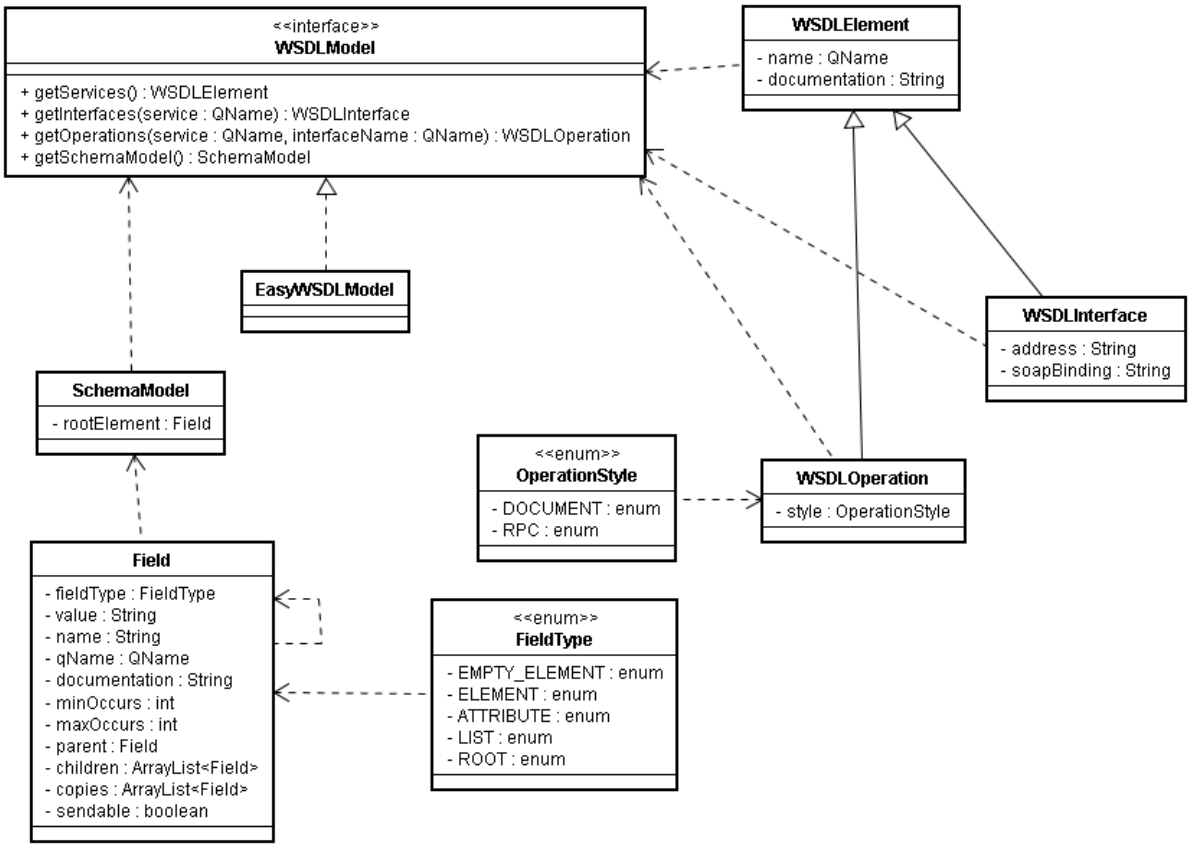


Figura 41 - Modelo de dados

A utilização deste modelo de dados inicia na interface WSDLModel, que representa o modelo de dados simplificado do documento WSDL. Esta interface é implementada pela classe EasyWSDLModel, de modo a utilizar todo o poder da biblioteca EasyWSDL para obter os elementos desejados do documento WSDL.

Os métodos da classe WSDLModel são responsáveis por extrair informações do WSDL de forma seqüencial. Para se obter a informação completa de como se constrói a mensagem de envio e de retorno é primeiramente necessário selecionar o serviço, logo em seguida a interface e por último a operação. Cada um dos elementos retornados por estes métodos, contem uma parte do conteúdo necessário para que a mensagem seja enviada com sucesso.

A partir do momento em que se selecionou o método desejado, pode-se obter as informações dos campos que futuramente deverão ser preenchidos pelo usuário, através do modelo de dados proveniente da classe SchemaModel.

A classe SchemaModel é o início da estrutura de dados que será utilizada pela InputView para renderização dos campos em tela. O XML Schema atrelado a um método do WSDL é o que descreve os campos que devem ser exibidos e de que forma. Esta classe contém somente um atributo que é o nodo principal do modelo XML Schema. Para uma mensagem RPC, este nodo é o nome do método que será invocado e para uma mensagem Document/Literal este nodo é o nome do tipo de dado que irá trafegar. Este nodo é representado pela classe Field.

A classe Field é um POJO (*Plain Old Java Object*) de grande importância para o projeto FUSION. A classe é construída no momento da montagem do modelo de dados e a partir de então trafega entre as camadas sendo alimentada conforme o fluxo de execução ocorre, até o momento de montagem da requisição da mensagem a ser enviada.

A partir de indicadores como fieldType e sendable, pode-se inferir como o campo deve ser renderizado em tela, enviado ao servidor ou até mesmo se deve ser ignorado na montagem de requisição, representando apenas um *label* em tela.

Tendo o objeto Field que representa o modelo de dados criado a próxima tarefa do software é renderizar os dados em tela, conforme o próximo capitulo aborda.

4.7.5 Montagem da tela

Conforme anteriormente visto, o objeto Field é o objeto que trafega entre as camadas da aplicação. Ao chegar na camada view, o mesmo contém as informações que devem ser renderizadas e alimentadas pelo usuário para que a mensagem SOAP seja montada.

Para tanto, a tela "InputView" recebe o objeto e imediatamente invoca a classe auxiliar FXConverter, responsável por transformar o modelo de dados do objeto Field em campos a serem exibidos ao usuário. Para tanto, a classe FXConverter executa um algoritmo recursivo que navega nos objetos filhos da classe Field, procurando por elementos recebidos do XML Schema que sejam importantes para serem exibidos em tela. Ao encontrar estes elementos, a classe invoca o método auxiliar "convertFieldToFX", que decide qual tipo de componente deve ser exibido para determinado elemento. Este fluxo pode ser visualizado na Figura 42.

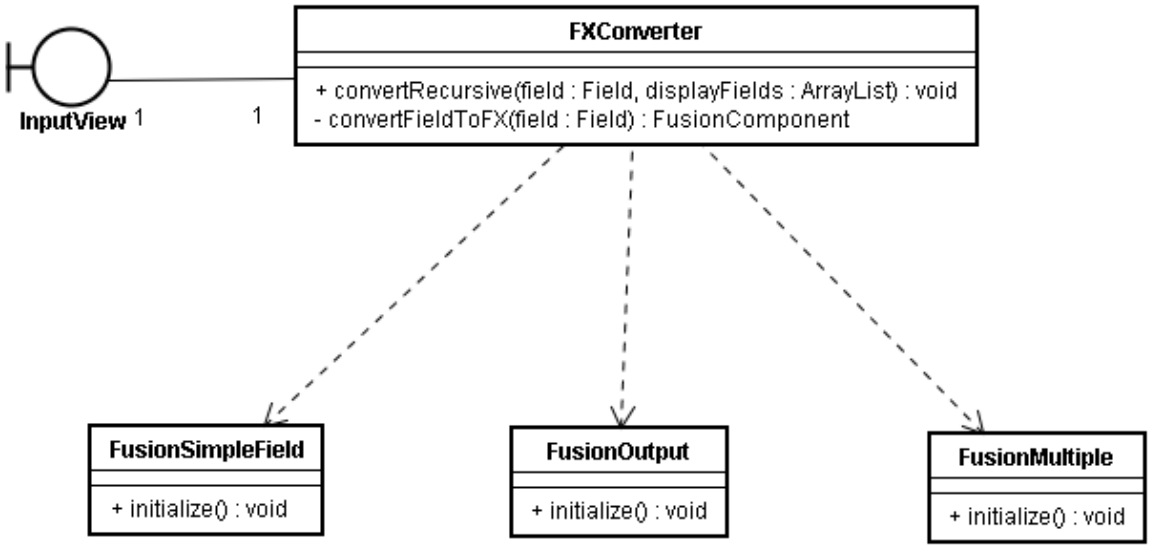


Figura 42 - Montagem dinâmica de campos

Uma característica importante do diagrama de classe apresentado na Figura 42 é que o objeto retornado do método de conversão de um Field para um campo de tela é um objeto do tipo FusionComponent, que é super classe dos componentes reais. Desta forma é possível expandir o produto, com tratamentos cada vez mais refinados para cada elemento encontrado em um XML Schema, sem impactar em grandes alterações de código.

JavaFX apresenta uma característica particularmente interessante para a atividade de exibição e preenchimento de campos dinamicamente criados. A funcionalidade de bind permite que neste momento, cada campo criado dinamicamente seja associado a variável value do objeto Field e a cada alteração de um campo de tela por parte do usuário, o valor da variável também passa a ser atualizada, evitando desta forma uma nova interação recursiva no objeto Field para alimentar os valores a serem enviados ao processo de montagem da requisição.

Após o preenchimento dos dados pelo usuário, o mesmo clica para que seja enviado a mensagem ao servidor e com isto é necessário converter os dados de tela em uma requisição HTTP. Este processo é explicado no próximo capítulo.

4.7.6 Montagem da requisição

No projeto FUSION, a montagem de uma requisição HTTP, consiste em transformar o objeto Field em um objeto do tipo java.xml.SOAPBody, que corresponde ao corpo da mensagem SOAP e seu respectivo conteúdo.

Outra requisito importante para a montagem da requisição é saber qual o modelo de mensagem SOAP esperada, podendo neste caso ser Document/literal ou RPC/literal.

Conforme a Figura 43 demonstra, o objeto WSConnection agrega tanto o objeto builder quando o produto retornado do builder.

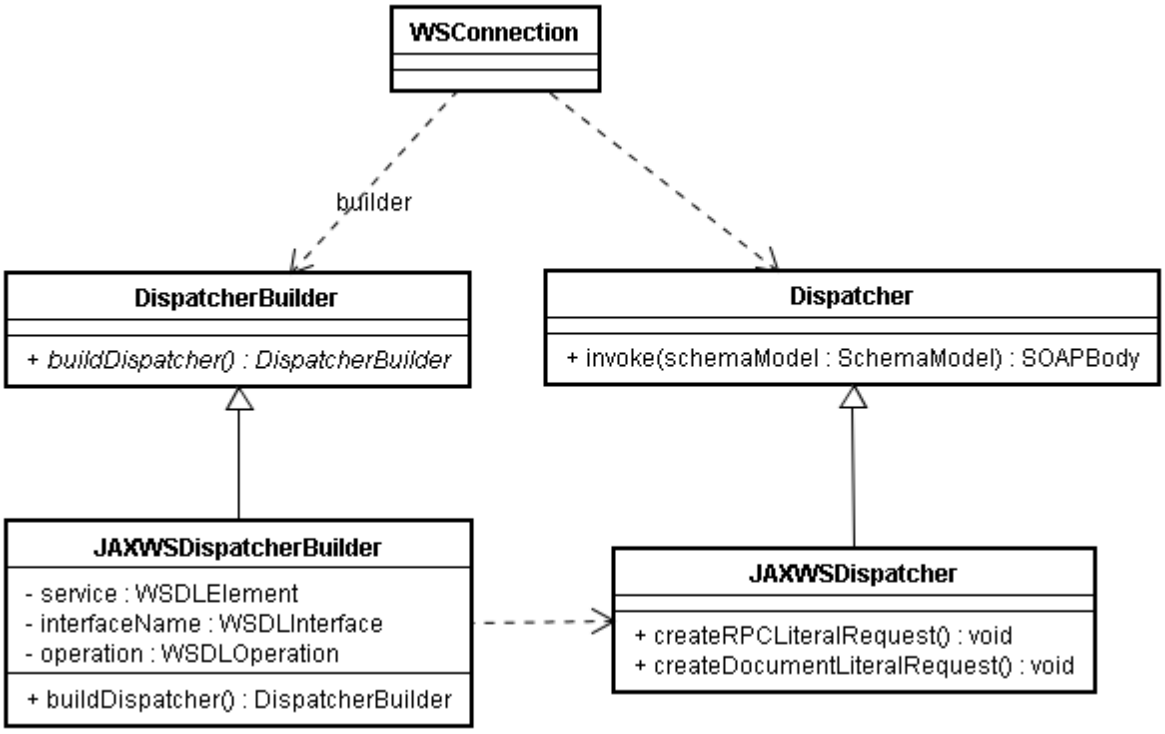


Figura 43 - Montagem da requisição

A construção de um objeto despachante de mensagem SOAP necessita de uma série de informações, motivo que levou a utilização do padrão de projeto Builder. Estas informações são preenchidas invocando os métodos set da classe de implementação, para logo em seguida construir o objeto complexo. O objeto construído é um objeto despachante que utiliza a tecnologia JAX-WS para envio de mensagens ao servidor.

Tendo o despachante construído com o modelo de dados correto, basta que seja enviado a informação do servidor para exibição do resultado.

Todo o procedimento apresentado em diagramas de classes pode ser visualizado em uma sequência de execução. Para facilitar a compreensão do trabalho, a próxima seção explica os principais pontos do projeto utilizando diagramas de sequência.

4.8 Diagrama de sequência

A partir dos diagramas de sequência será possível entender como o projeto FUSION processa cada uma das situações envolvendo uma conexão dinâmica a um *Web Service*. O primeiro diagrama, demonstrado na Figura 44, demonstra como a partir da tela de escolha de serviço o software permite a escolha dos dados disponíveis no serviço.

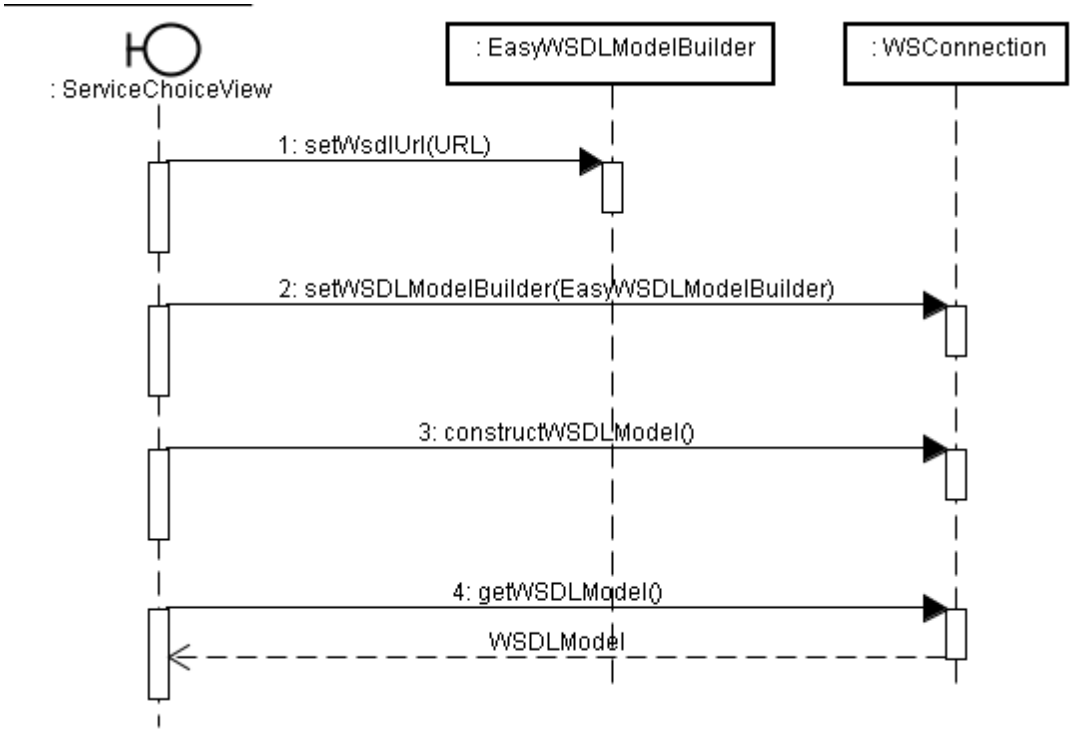


Figura 44 - Sequência de construção do objeto WSDLModel

Novamente pode-se encontrar a presença do padrão de projeto *Builder*, uma vez que para se obter o modelo de dados do *Web Service* foi necessário alimentar o objeto EasyWSDLModelBuilder e atribuí-lo de parâmetro para a classe WSConnection construí-lo.

Após a execução desta sequência, os componentes de seleção das opções do serviço estão disponíveis para utilização. Neste momento é necessário que a partir das opções selecionadas pelo usuário seja retornado o modelo de dados do *XML Schema*. Este modelo será exibido na tela de entrada de dados da mensagem a ser enviada (*InputView*). O processo de seleção de opções pode ser visualizado no diagrama da Figura 45.

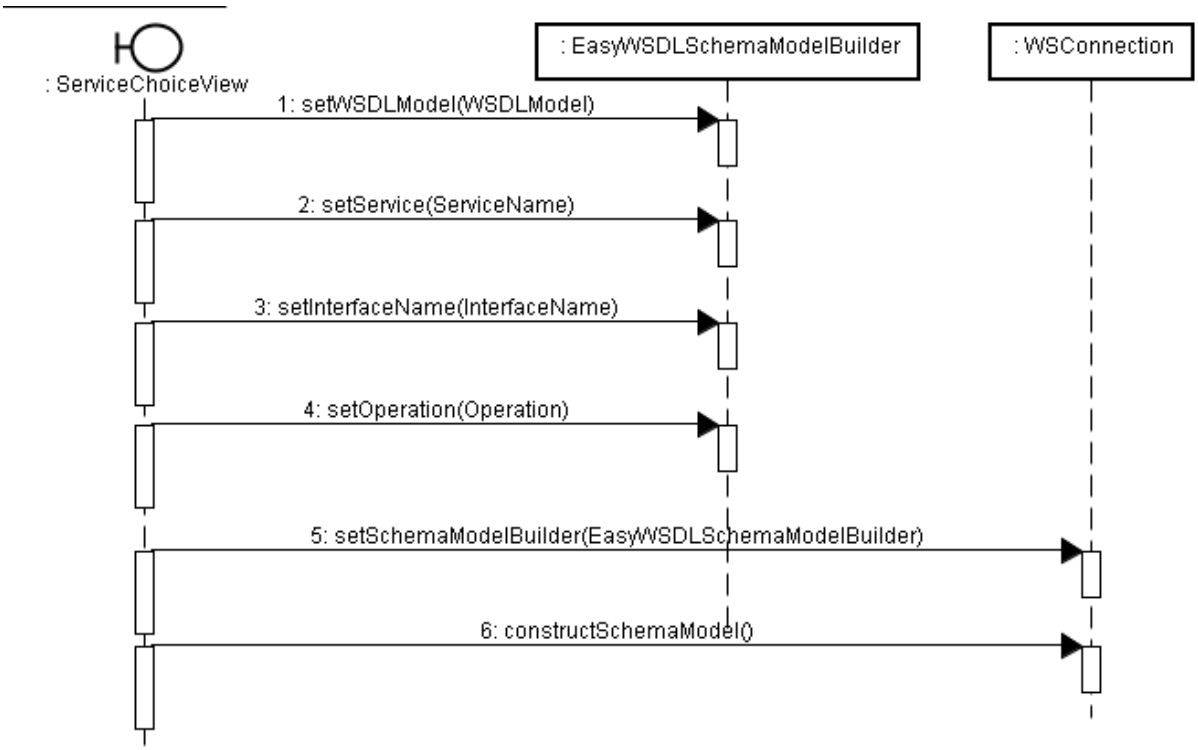


Figura 45 - Sequência de construção do objeto SchemaModel

Para se obter o modelo de dados do *XML Schema*, foi necessário que um dos argumentos informados fosse o modelo de dados do WSDL, uma vez que o *Schema* está atrelado ao WSDL.

A partir deste momento, o software já está pronto para renderizar a tela de entrada de dados. Porém, assim que o objeto SchemaModel é construído, o objeto despachante da mensagem também é criado, com o objetivo de centralizar a regra de negócio envolvida na manipulação de *Web Services* em um único ponto. A construção do objeto despachante pode ser visualizada na Figura 46.

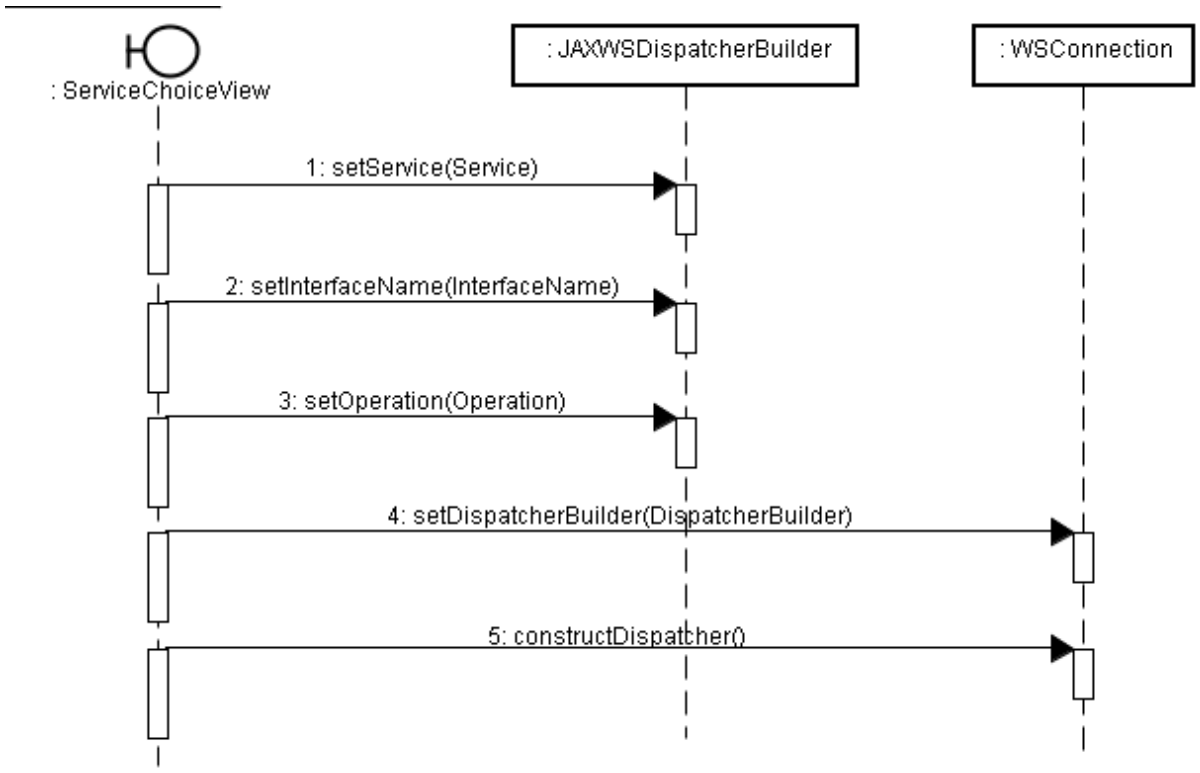


Figura 46 - Sequência de construção do objeto Dispatcher

O produto de todas os diagramas de sequência apresentados anteriormente se encontra agora no objeto WSConnection. Este objeto agora é enviado a tela de entrada de dados, para que seja montada a interface dinâmica. O fluxo de montagem da tela pode ser visualizado na Figura 47.

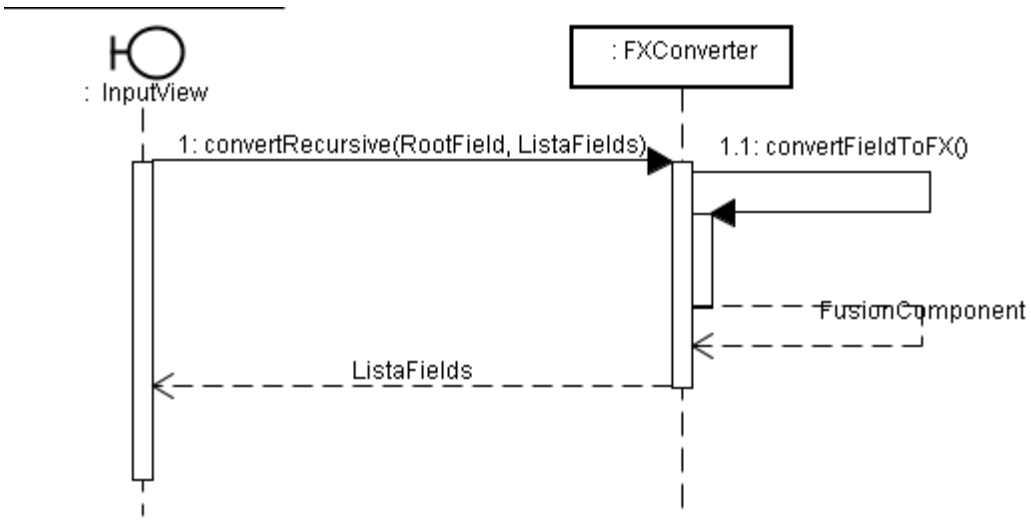


Figura 47 - Sequência para montagem da tela dinâmica

Os parâmetros do método “convertRecursive” são conforme a Figura 47, um objeto do tipo Field, que é obtido a partir do do SchemaModel e uma lista de campos vazia, que na implementação é um objeto do tipo java.util.ArrayList. Esta lista vazia será alimentada por componentes de tela a serem exibidos.

Tendo a tela com os campos renderizados e preenchidos pelo usuário, o ultimo fluxo a ser executado é o de envio da mensagem ao *Web Service* destino, conforme a Figura 48 ilustra.

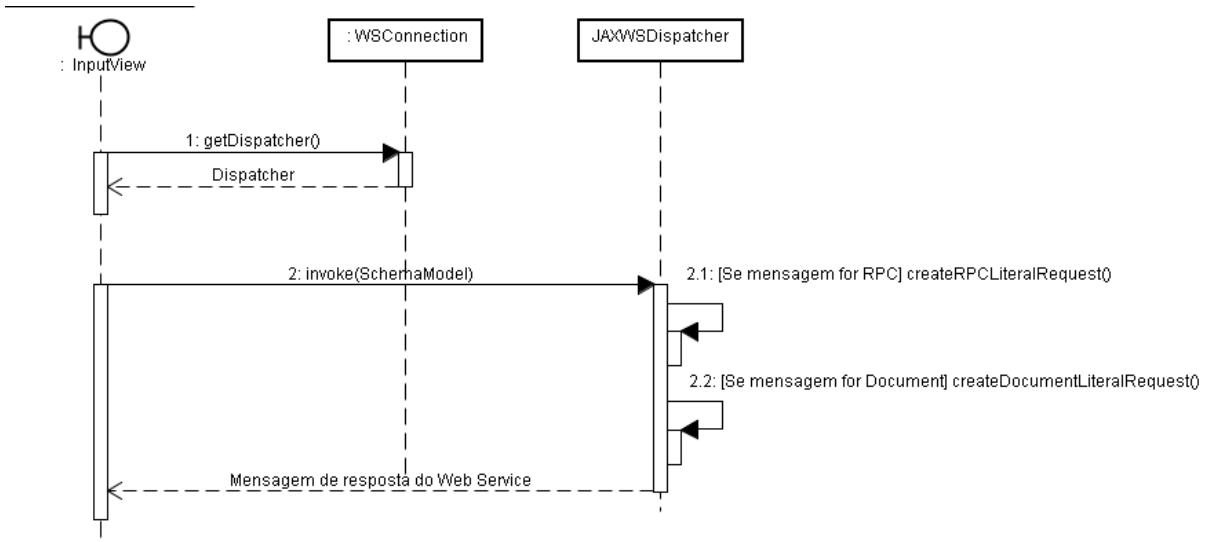


Figura 48 - Sequência para enviar a mensagem ao Web Service

Após enviar a mensagem, basta apresentar a resposta na tela de exibição de resultados (ResultView), representado o término do fluxo de execução do software.

No próximo capítulo serão apresentadas telas do software, para que seja possível criar uma melhor analogia entre os diagramas e fluxos apresentados com o produto gerado.

4.8 Produto desenvolvido

O projeto FUSION foi concebido com a idéia de ter uma interface gráfica simples, com botões significativos e com formas arredondadas. A Figura 49 apresenta a tela inicial do projeto.

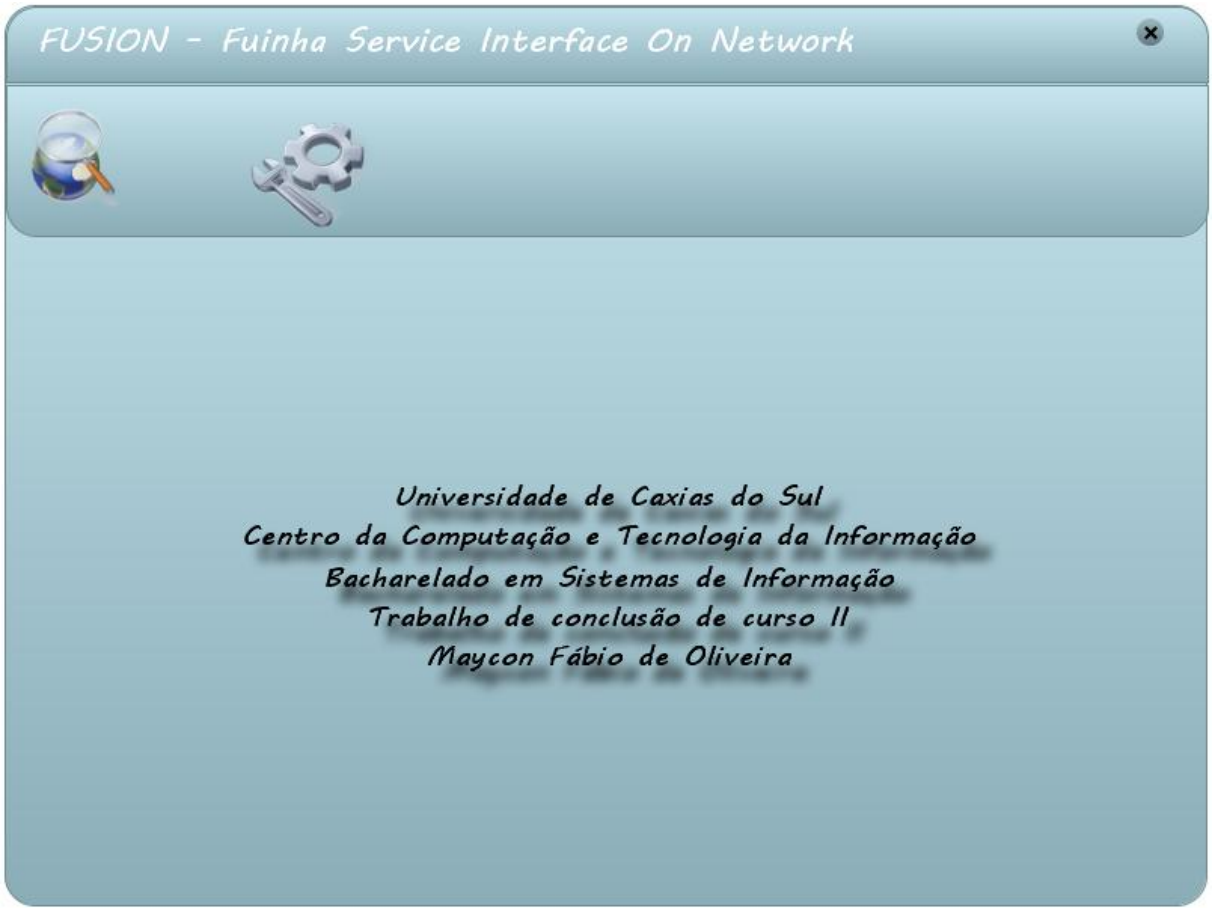


Figura 49 - Tela inicial do sistema (MainView.fx)

Através da alteração do estilo do objeto Stage do JavaFX, é possível criar uma aplicação que não utilize a barra de título padrão dos sistemas operacionais tais como Windows e Linux que oferecem por default os botões minimizar, maximizar e fechar.

A tela da Figura 49 é composta por um leiaute criado na Main.fx e pela view MainView que exibe os dados do projeto e da instituição a qual se destina.

A partir desta tela está disponível através de um clique no ícone do globo a chamada da tela de seleção do serviço desejado dentre os oferecidos pelo WSDL definido em uma URL, conforme a Figura 50 demonstra.

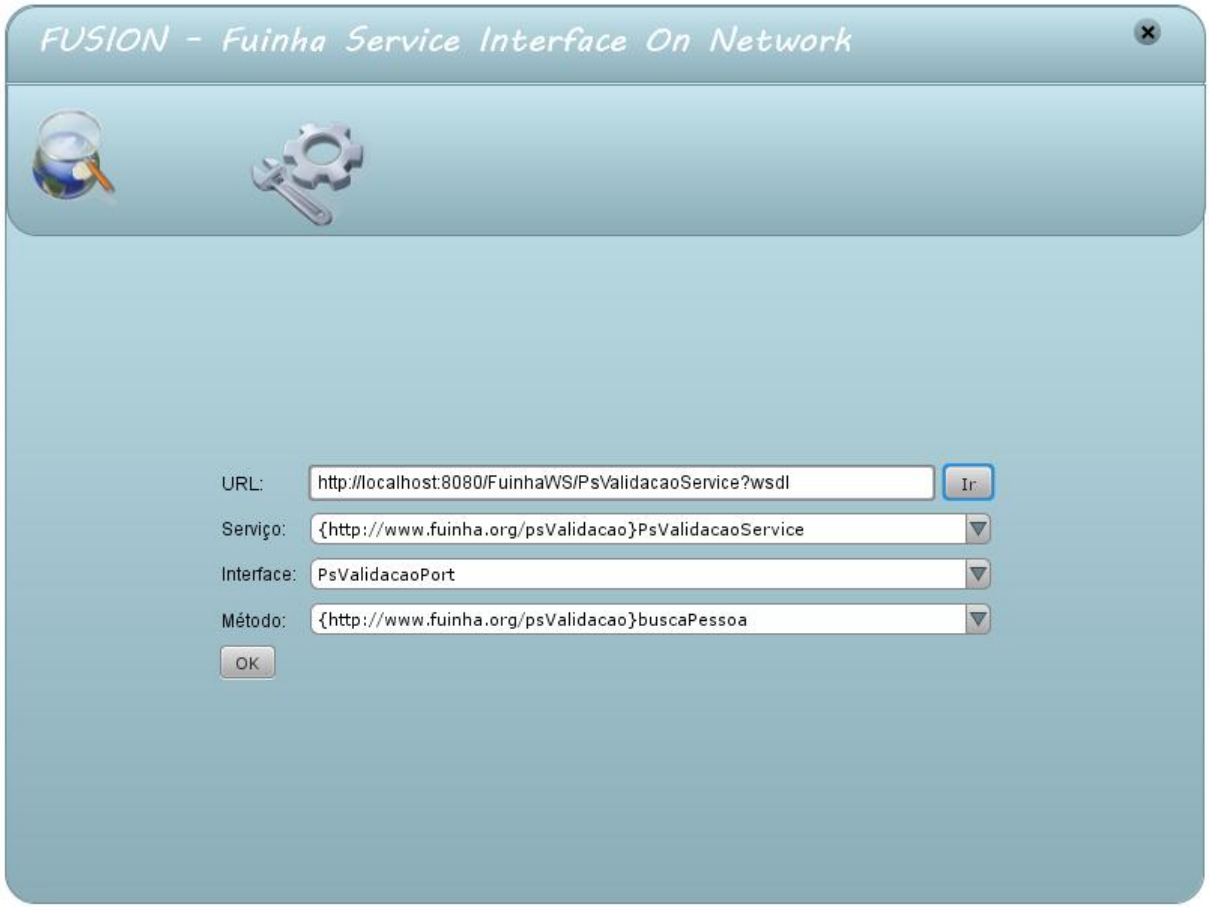


Figura 50 - Tela de seleção de *Web Service* (ServiceChoiceView.fx)

A Figura 50 representa a tela de seleção de serviço implementada na classe ServiceChoiceView.fx. Nesta tela é possível através da digitação da URL desejada, liberar as opções disponíveis, selecionar o serviço, a interface e o método desejado.

Com a combinação selecionada pelo usuário, é possível extrair os dados necessários para a montagem da tela dinâmica, resultando em uma tela semelhante à demonstrada na Figura 51.

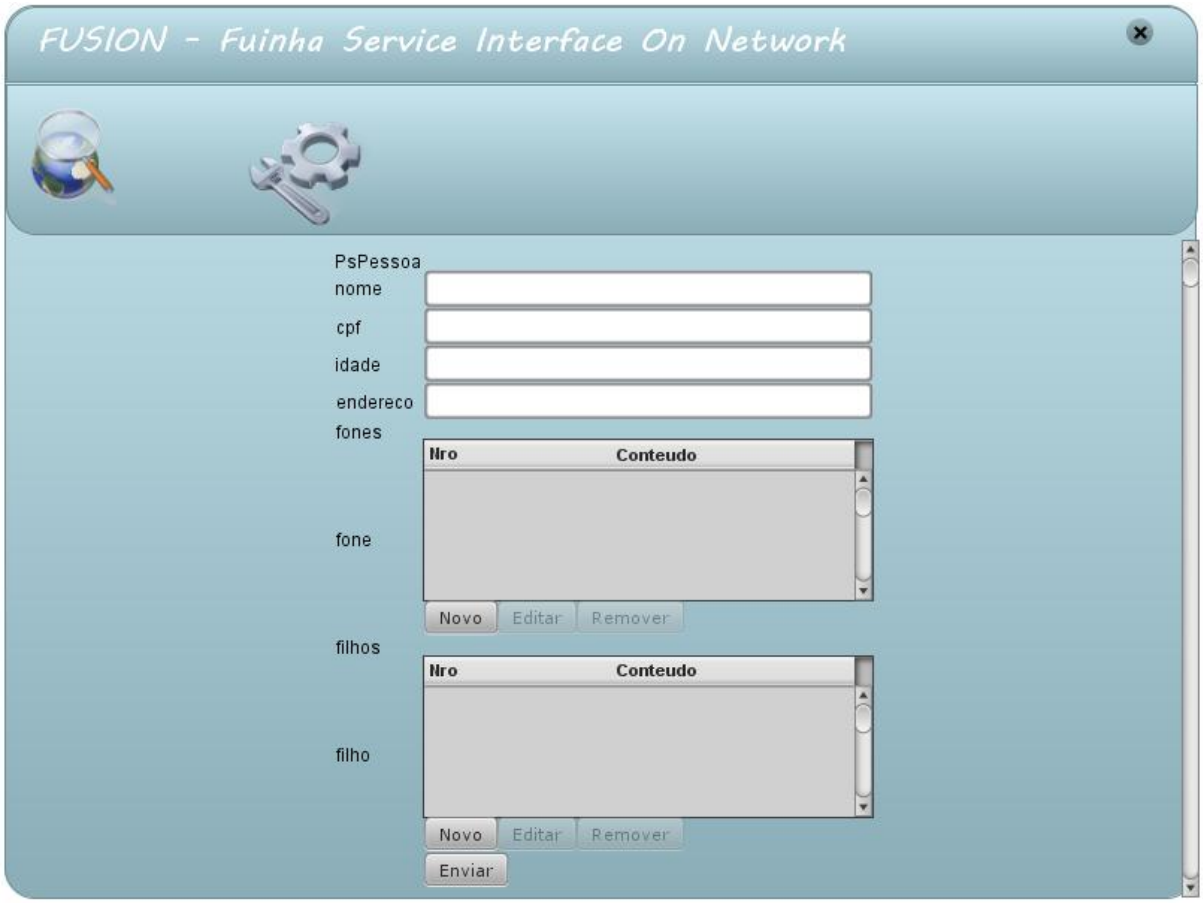


Figura 51 - Tela de entrada de dados (InputView.fx)

A tela de entrada de dados é a tela que apresenta a maior quantidade de componentes customizados para a aplicação. Na Figura 51 é possível visualizar os componentes FusionOutput (*label* simples), FusionSimpleField (campos textos) e FusionMultiple (listagens).

Uma característica importante de um software dinâmico para *Web Service* é a apresentação dos rótulos. Cada *label* apresentado ao lado esquerdo da tela representa uma *tag* XML processada, razão a qual os *labels* não contêm acentuação.

A mesma tela da Figura 50, durante a entrada de dados pode ser visualizada como a Figura 52.

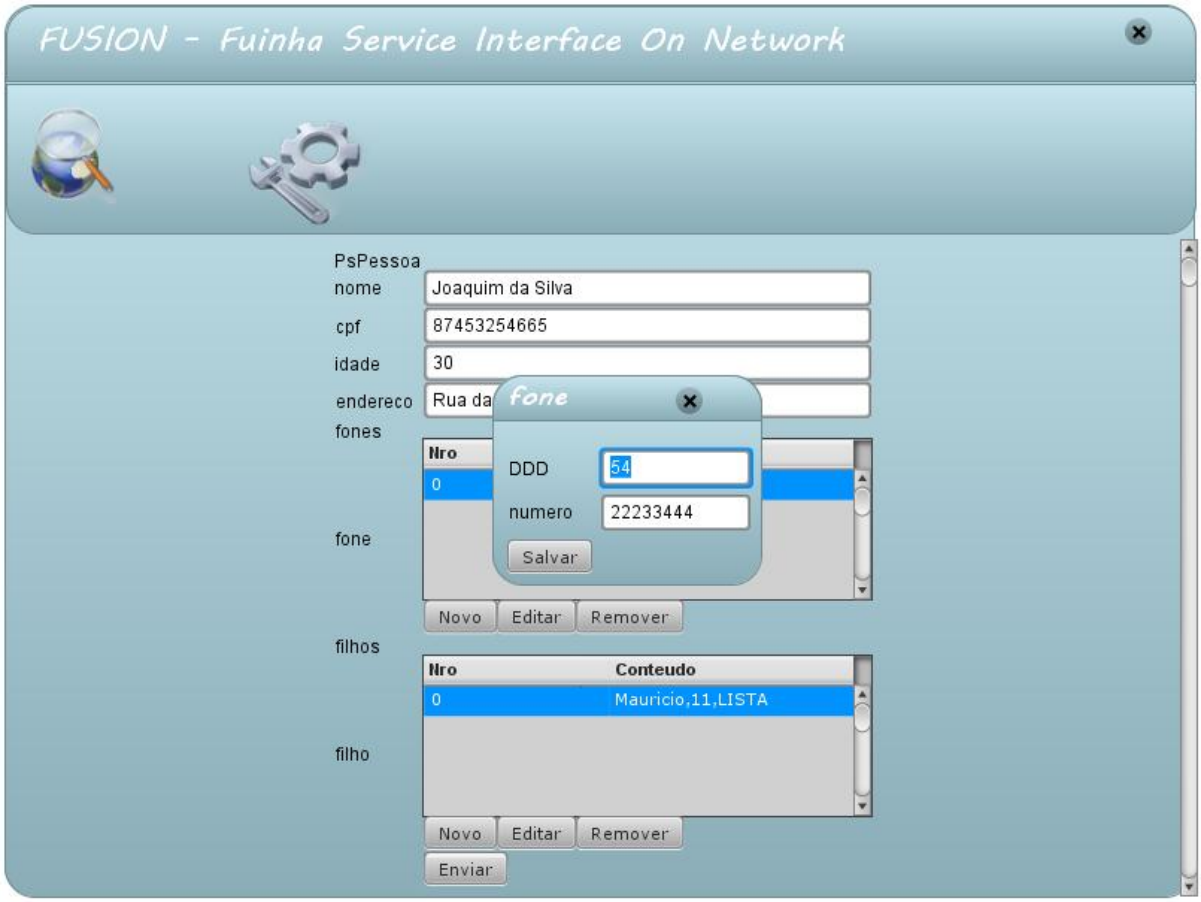


Figura 52 - Entrada de dados

Na Figura 52 também é possível verificar mais uma parte do componente de listagem que é a janela de diálogo aberta quando um item é criado ou editado.

Após o envio da mensagem ao servidor através do botão “Enviar”, a última tela do fluxo de comunicação com o *Web Service* é exibida, conforme a Figura 53 ilustra.

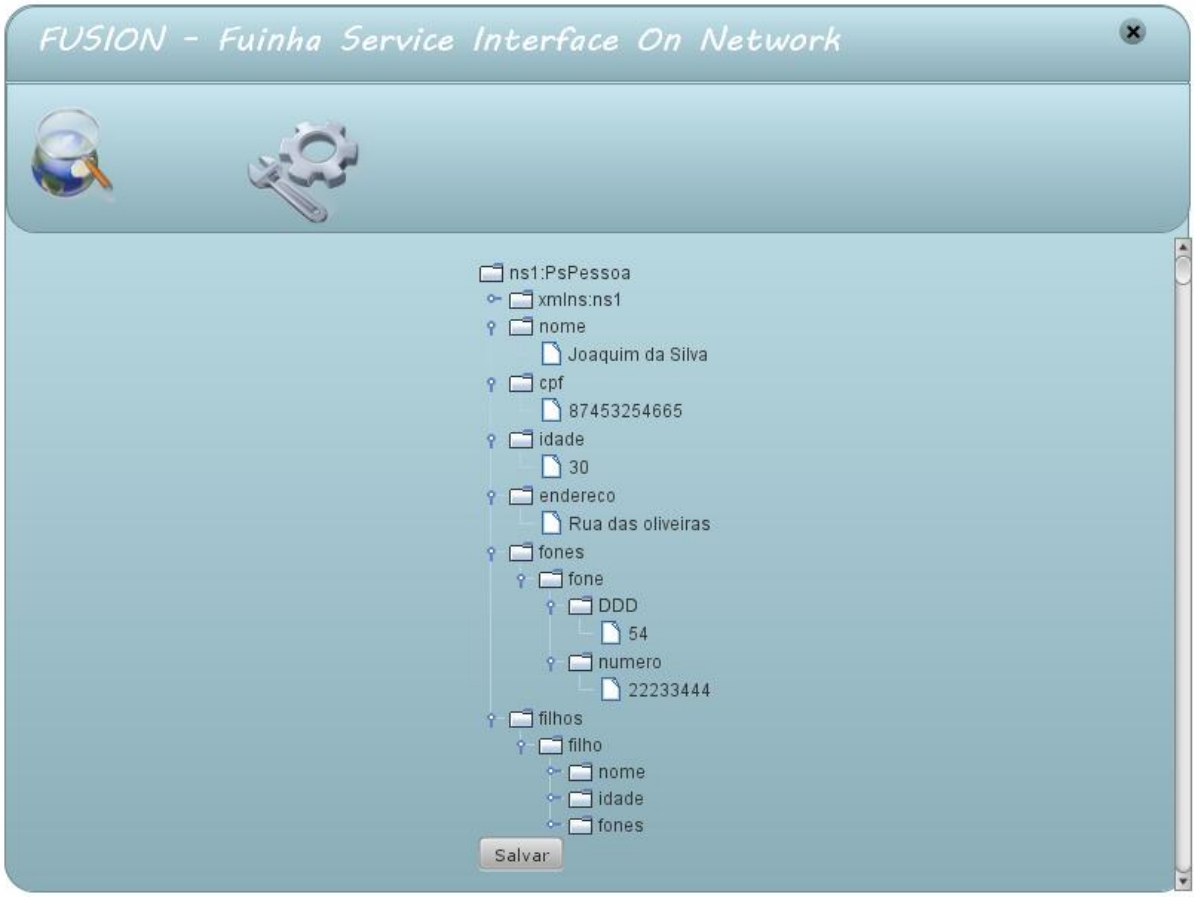


Figura 53 - Tela de exibição do resultado (ResultView.fx)

A tela da Figura 53 cria uma árvore com o XML retornado do servidor, neste caso, informações sobre o usuário cadastrado anteriormente. Também está disponível nesta tela a opção de salvar o XML da árvore em disco.

Até este momento foi apresentado o software criado, suas funcionalidades e sua interface gráfica. Porém, uma grande estrutura de testes foi utilizada para que a construção do presente trabalho fosse possível. Esta estrutura será explicada na seção a seguir.

4.9 Estrutura de testes

Para validar um software de comunicação dinâmica, a estrutura de testes deve ser capaz de simular algumas das diversas situações a qual software poderá ser solicitado a interagir.

Para tanto, durante o desenvolvimento, diversos *Web Services* foram criados, alguns deixando que o próprio servidor de aplicação se responsabilizasse em gerar o documento WSDL (abordagem *bottom-up*) e outros criados a partir do documento WSDL (abordagem *top-down*).

Utilizando a abordagem top-down, foi realizado também a implementação do serviço de solicitação de procedimentos médicos, disponibilizado no projeto TISS da Agência Nacional de Saúde. Este *Web Service* necessita de uma grande quantidade de informações de diversos tipos de dados, que devem ser enviadas por parte da aplicação cliente para que uma mensagem seja aceita. Este serviço além de testar a capacidade de processamento da aplicação, testa também os algoritmos recursivos e valida o projeto contra uma situação real de utilização do produto.

Para construção dos *Web Services* foi utilizado as seguintes ferramentas:

- Eclipse: IDE de desenvolvimento;
- JBoss 4.2.3: Servidor de aplicação;
- JBoss Tools: Plugin para a IDE eclipse que permite a integração do servidor de aplicação JBoss com a IDE.
- Apache Ant: Linguagem de script inicialmente utilizada para construção de aplicações Java. No projeto a mesma foi utilizada para gerar classes a partir do documento WSDL.

Devido o foco do trabalho ser a criação de um cliente de *Web Services*, o assunto não será aprofundado sobre o ambiente de testes, porém, para gerar um *Web Service* a partir da abordagem *top-down* no ambiente criado, os seguintes passos devem ser realizados:

- Criar um documento WSDL;
- Criar uma nova entrada no script do Ant que aponte para o WSDL criado;
- Criar um *Session Bean* que implemente a interface gerada pelo Ant;
- Adicionar anotações no *Session Bean* para referenciar ao WSDL criado.

Todos os passos acima citados podem ser reproduzidos realizando uma cópia dos artefatos já existentes no projeto FuinhaWSImpl, em anexo a este trabalho.

No próximo capítulo serão apresentadas algumas considerações sobre o trabalho desenvolvido, desde o momento de sua concepção.

5 CONSIDERAÇÕES SOBRE O PROJETO

A proposta da construção de um cliente de *Web Service* dinâmico, apesar de perfeitamente viável, despertava preocupações, pois a quantidade de situações e tecnologias sob a qual o software iria ser desenvolvido era relativamente grande. Porém, esta dificuldade foi gradativamente superada, graças a um grande estudo em sites de referência das linguagens envolvidas e também a aplicação de padrões de projeto que abstraiam a complexidade do software. Porém, o projeto FUSION enfrentou desde o momento da concepção e dos testes preliminares, uma grande freqüência de atualizações de algumas tecnologias e baixa freqüência por parte de outras.

A primeira tentativa de desenvolvimento do projeto ocorreu utilizando a IDE Eclipse, por uma questão de conhecimento prévio deste ambiente por parte do autor. Nesta IDE foram testados 2 *plugins* para desenvolvimento de aplicações JavaFX, um disponibilizado pela SUN e outro pela Exadel. Em ambos a IDE se comportou mal, corrompendo arquivos, perdendo referências, além da não oferecer uma ferramenta de depuração ou até mesmo identação de código.

Partiu-se então para a utilização da IDE NetBeans 6.7.1 que disponibilizava um ambiente completo de desenvolvimento para a tecnologia JavaFX 1.2. Alguns meses depois, a aplicação foi migrada para a versão 6.8 desta IDE.

O projeto FUSION depende de dois conjuntos de bibliotecas auxiliares, o JFXTras e o EasyWSDL. Estas bibliotecas ativeram suas versões atualizadas diversas vezes durante o desenvolvimento do projeto e as alterações realizadas não eram retro-compatíveis, disparando exceções em trechos do código já devidamente escritos e testados. Estas bibliotecas também possuem uma documentação bastante fraca, obrigando que o desenvolvimento do projeto fossem baseado em testes unitários e pesquisas na internet.

Outro problema que se apresentou foi o atraso na liberação da versão 1.3 do JavaFX. O desenvolvimento da interface gráfica aguardava desde a elaboração da proposta do trabalho pela nova liberação da linguagem, que traria consigo grande parte dos componentes inexistentes na versão 1.2, tais como janelas de dialogo, combo-boxes, etc. Por demora da liberação, decidiu-se por iniciar o desenvolvimento na versão 1.2. A liberação da 1.3 ocorreu no dia 22/04/2010, inviabilizando uma migração devido ao estágio avançado do projeto e da necessidade de entrega de outros artefatos junto ao mesmo. Para que esta migração ocorresse também seria necessário aguardar pela liberação da nova versão do JFXtras para JavaFX 1.3,

além de atualizar o código do projeto para substituir métodos que deixaram de existir na nova API.

O último problema enfrentado é relativo ao grande número de pontos de recursividade apresentados pelo software e a depuração do código em uma situação como esta. Ao todo foram aplicados quatro pontos de recursividade, nas seguintes situações:

- Durante a interpretação do documento WSDL;
- Durante a montagem da interface;
- Durante a montagem da requisição de envio;
- Durante a montagem da árvore da tela de resultado;

Apesar de todos os problemas apresentados o projeto obteve sucesso em relação à proposta realizada.

6 CONCLUSÃO

Conforme pôde ser visualizado no presente trabalho, *Web Service* é uma tecnologia bastante abrangente que possibilita sua utilização para diversas finalidades, dependendo da forma como se monta o serviço destino e o cliente que se comunica com o serviço.

Muito além de simplesmente se comunicar com *Web Service*, o presente trabalho demonstra que é possível a construção de um cliente dinâmico, que se adapte ao serviço desejado, ao mesmo tempo em que expõe uma interface gráfica para interação.

Duas grandes sugestões de aplicação e de continuidade do trabalho podem ser formuladas após a leitura do trabalho. A primeira diz respeito à atualização do JavaFX para a última versão disponível. A cada release mais componentes são liberados, a performance melhora e a linguagem se fortalece a medida em que mais ferramentas são criadas para desenvolvimento. Ao partir para esta meta, pode-se melhorar a interação do usuário com o software, ao mesmo tempo em que se aposta em uma linguagem relativamente recente.

A segunda sugestão está relacionada com ao acréscimo de regras de interpretação de dados do serviço. Inicialmente pode-se através das informações nativas oferecidas pelo XML Schema, melhorar a interface acrescentando funcionalidades como validação de tamanho máximo de campos, máscaras de formatação, entre outras. Pode-se também avançar e utilizar *tags* especiais do XML Schema como a appInfo e documentation que permite ao desenvolvedor de serviços, acrescentar metadados especiais para interpretação do cliente. Através de uma boa estrutura de metadados, seria possível realizar tarefas como validar interdependência de campos e orientar o software cliente a buscar informações em outras fontes de dados.

Além de simplesmente apresentar um cliente de *Web Services*, o trabalho também procurou, através da apresentação de uma abordagem diferenciada para utilização de uma tecnologia, despertar a atenção do leitor sobre a possibilidade de expandir a utilização de recursos existentes em prol de uma melhor eficácia na execução de seus trabalhos.

7 REFERÊNCIAS

ANDERSON, Gail, ANDERSON, Paul. **Essential JavaFX**. USA, Pearson Education, 2009.

ANS – **Agência Nacional de Saúde.** Disponível em http://www.ans.gov.br . Acesso em 29/03/2010.

BRADLEY, Neil. **The XML Companion.** USA, Addison Wesley, 2002.

BUTEK, Russel. **Which style of WSDL should I use?.** Disponível em http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/ . Acesso em 20/10/2009.

CLARKE, Jim, CONNORS, Jim, BRUNO, Eric. **JavaFX: Developing Rich Internet Applications**. USA, Addison Wesley, 2009.

CRUZ, Tadeu. **BPM & BPMS: Business Process Management & Business Management Systems**. Rio de Janeiro, Brasport 2008.

DOEDERLEIN, Osvaldo P. **JavaFX Script, uma nova definição para "linguagem dinâmica"**. Revista Java Magazine, ano VII, edição 69, 2009.

DOEDERLEIN, Osvaldo P. **Uma aplicação comum em JavaFX**. Revista Java Magazine, ano VII, edição 72, 2009.

ENGLANDER, Robert. **Java and SOAP.** USA, O´Reilly, 2002.

GANDOLPHO, Cibele. **Cresce uso de SOA.** Disponível em http://info.abril.com.br/corporate/infraestrutura/cresce-uso-de-soa.shtml Acesso em 10/03/2010.

HANSEN, Mark D. **SOA Using Java Web Services.** USA, Pearson Education, 2007.

HOHPE, Gregor, WOOLF, Bobby. **Enterprise Integration Patterns**. USA, Addison Wesley, 2003.

LINTHICUM, David S. **Enterprise Application Integration**. USA, Addison Wesley, 1999.

MARTINS, Victor Manuel Moreira. **Integração de Sistemas de Informação: Perspectivas, normas e abordagens**. Guimarães, 2005. Dissertação de Mestrado apresentada no Programa de Pós-Graduação em Sistemas de Informação da Universidade do Minho, Portugal.

MERSKER, Steven J. **Design Pattern Java Workbook. USA,** Addison Wesley, 2002.

MONSON-HAEFEL, Richard. **J2EE Web Services**. USA, Pearson Education, 2004.

MSDN – **Microsoft Developer Network**. Disponível em http://msdn.microsoft.com/ . Acesso em 15/03/2010.

NETO, Jorge Abílio Albinader, LINS, Rafael Dueire, **Web Services em Java**. Rio de Janeiro, Brasport 2006.

ROSENBERG, Doug, STEPHENS, Matt e COLLINS-COPE, Mark. 2005. **Agile Development with ICONIX Process: People, Process, and Pragmatism**. New York, Apress, 2005.

SAMPAIO, Cleuton. **SOA e Web Services em Java**. Rio de Janeiro, Brasport 2006.

W3C – **The World Wide Web Consortium**. Disponível em http://www.w3.org/ . Acesso em 18/09/2009.

SEFAZ – **Secretaria da Fazenda.** Disponível em http://www.nfe.fazenda.gov.br . Acesso em 29/03/2010.

SILVA, Alberto Manuel Rodrigues da, VIDEIRA Carlos Alberto Escaleira **UML, Metodologias e Ferramentas CASE.** Lisboa, Centro Atlântico, 2001.

TERZIAN, Françoise, CORRÊA Ricardo. **Prepare-se para a era dos Web Services.** Disponível em http://info.abril.com.br/canal/edicoes/9/conteudo_133803.shtml. Acesso em 10/03/2010.

TYAGY, Sameer. **Patterns and Strategies for Building Document-based Web Services**. Disponível em http://java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns/. Acesso em 16/03/2010.

WS-I – **The Web Service Interoperability Organization.** Disponível em www.ws-i.org . Acesso em 20/10/2009.