Lourival F. de Almeida Júnior

Explorando Teste Baseado em Perturbação no Contexto de *Web Services*

Orientadora: Profa. Dra. Silvia Regina Vergílio

CURITIBA
2006

Lourival F. de Almeida Júnior

Explorando Teste Baseado em Perturbação no Contexto de *Web Services*

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Profa. Dra. Silvia Regina Vergílio

Universidade Federal do Paraná

CURITIBA
2006

Resumo

Web Service é uma tecnologia moderna comumente utilizada para integrar projetos de *software* de diferentes plataformas, sistemas operacionais ou mesmo linguagens de programação. Esta natureza distribuída e heterogênea dificulta a atividade de teste, que em geral, é bastante custosa e requer grande esforço para ser completada, logo, métodos mais adequados e menos custosos de teste são necessários para *Web Services*. Uma abordagem baseada em perturbação de mensagens XML foi introduzida para testar pares de *Web Services*. Este trabalho explora a utilização desta abordagem introduzindo novos operadores de perturbação para mensagens SOAP. Os operadores de perturbação são responsáveis por produzirem mensagens modificadas que serão utilizadas como casos de teste para *Web Services*. Uma ferramenta, chamada SMAT-WS, para dar apoio à aplicação destes operadores foi implementada. Com esta ferramenta foi realizado um experimento que indicou a viabilidade de aplicação dos operadores de perturbação no contexto de *Web Services*. A aplicação destes operadores se torna mais eficaz em momentos mais iniciais do desenvolvimento destes serviços.

Abstract

Web Service is a modern technology commonly used to integrate software projects among different platforms, operating systems or even programming languages. This distributed and heterogeneous nature complicates the testing activity which is, in general, expensive and effort demanding. Adequate and cost effective testing approaches are needed for Web Services. An approach based on XML message perturbation has been introduced to test pairs of Web Services. Perturbation operators produce modified SOAP messages, which are used as test cases. This work explores the use of such promising approach by introducing new perturbation operators for SOAP messages and describing a supporting tool, named SMAT-WS. With this tool an experimental study was accomplished. The obtained results have shown that the application of perturbation operators is practicable, specially when applied to initial moments of a projetct development life cycle.

Sumário

Lista de Figuras

Lista de Tabelas

"Qualquer atividade torna-se criativa quando
quem a executa se preocupa em
fazê-la direito ou melhor".
John Updike

1. Introdução

A Internet vem se consolidando como um meio de comunicação indispensável e popular. Muitas pessoas, empresas, bancos, instituições de ensino, entre outros, utilizam-se de suas facilidades para fazer negócios ou mesmo trocar informações confidenciais entre si. Desta forma, problemas em aplicações baseadas na *web* ou aplicações distribuídas podem causar conseqüências de altas proporções [WUO02].

Cada vez mais empresas não só utilizam aplicações baseadas na *web* para se posicionar na Internet, como se tornam mais dependentes destas aplicações. As soluções baseadas na *web* ou aplicações de *e-business* são parte integrante da infra-estrutura de Tecnologia da Informação de muitas corporações, permitindo a interligação de empresas clientes e parceiras por meio da Internet.

No entanto, a Internet e toda sua quantidade de informações foram idealizadas e construídas para ser interpretada por seres humanos, ou seja, os dados contidos em cada página são de difícil interpretação por *software* [REY06].

Neste contexto, os *Web Services*, definidos como programas que operam de forma independente para fornecer serviços na Internet para outros programas, incluindo aplicações *web* e outros *Web Services* [XUO05], podem ser utilizados para criar novas oportunidades de negócio ou mesmo agregar valor a serviços de forma a ganhar vantagem competitiva sobre os concorrentes de uma determinada empresa, uma vez que os dados que trafegam entre eles são prontamente entendidos por outros componentes de *software*.

Com o aumento na utilização da tecnologia de *Web Services* algumas preocupações são introduzidas e se tornam essenciais para sua completa aceitação, a saber:

- Garantia de disponibilidade destes serviços;
- Segurança da informação oferecida com a utilização destes serviços;
- Interoperabilidade entre os mais diversos serviços de diferentes desenvolvedores;
- Distribuição destes serviços;
- Corretude das funcionalidades oferecidas por estes serviços.

Ainda, um dos maiores desafios para a construção de *software* baseado na *web* é que seus requisitos exigem diversos atributos de qualidade implementados, como confiabilidade, usabilidade, manutenibilidade, entre outros [OFF02]. Contudo, a maior parte da indústria de *software* tem obtido sucesso com relativamente poucos requisitos de qualidade implementados, o que é explicado por uma combinação de estratégias de mercado e *time-to-market*. Com isso, as produtoras de *software* convencional ou tradicional têm conseguido colocar no mercado produtos competitivos e de sucesso. No entanto, para aplicações baseadas na *web*, esta lealdade do consumidor aparentemente não existe, o que significa que, se uma empresa A publica um *site* na *web* antecipadamente e B entra no mercado posteriormente, se o usuário percebe que o *site* da empresa B possui um nível de qualidade mais elevado, então este usuário rapidamente mudará para o *site* da empresa B. Ou seja, na área de aplicações *web*, normalmente, vale estar atrasado mas de melhor qualidade do que se antecipar ao mercado publicando um produto de qualidade inferior, inadequada ou mesmo questionável [WUO02]. Pode-se notar então que os requisitos de qualidade são normalmente tão importantes quanto os demais requisitos, quando se trata de aplicações baseadas na *web*.

Como uma atividade diretamente ligada à qualidade, a atividade de teste tem como um de seus principais resultados a aferição do nível de confiabilidade do *software* em avaliação [PRES02]. No entanto, os métodos utilizados para testar aplicações tradicionais e aplicações *web* diferem devido às características inerentes a cada tipo de aplicação. As aplicações baseadas na *web*, por exemplo, devido aos seus requisitos que exigem alta qualidade, como comentado anteriormente, justificam a necessidade de desenvolvimento de tecnologias de teste diferenciadas para este tipo de aplicação. Tais tecnologias devem ser criadas de forma a apoiar a utilização de ferramentas automatizadas capazes de reduzir o esforço requerido pelos envolvidos nos procedimentos de teste deste tipo de *software*, assim como

acontece com aplicações tradicionais. A tecnologia de *Web Services*, por integrar as características das aplicações *web* além da heterogeneidade e o fato de ser uma tecnologia distribuída, ainda é considerada um grande desafio para a disciplina de testes [DIL05], fazendo-se necessário o desenvolvimento de métodos inovadores de verificação e validação para este tipo de aplicação.

Muito já tem sido estudado e proposto no que diz respeito a teste para aplicações tradicionais e para as arquiteturas cliente/servidor, porém, na área de aplicações distribuídas, mais especificamente para *Web Services*, tais técnicas não têm se mostrado tão eficientes para uso nestes contextos, uma vez que apresentam características bem peculiares, já citadas anteriormente. Contudo, apesar de toda a atenção que os *Web Services* vêm recebendo pela comunidade de Tecnologia da Informação, o número de fontes de informações sobre testes desta tecnologia é bastante reduzido [BLO02b].

Lee *et al.* [LEE01] descrevem um método, baseado no critério de Análise de Mutantes [DEM78], para testar aplicações baseadas na *web* que trocam mensagens entre si no formato XML. É apresentado um Modelo de Especificação de Interação (*Interaction Specification Model* – ISM), utilizado para gerar as mutações. Neste caso, os resultados destas mutações não serão novas versões de código, mas sim, novos dados que serão passados na troca de informações entre os componentes envolvidos.

Offutt e Xu [OFF04] introduziram uma abordagem baseada na perturbação de dados para testar *Web Services*. A idéia é modificar as mensagens de requisição com a utilização de operadores de mutação, assemelhando-se ao teste de mutação [DEM78], contudo, os mutantes gerados são na realidade os casos de teste a serem usados no teste de *Web Services.* Mensagens XML são modificadas baseadas em regras definidas na própria gramática, e então utilizadas como casos de teste. Foram propostos dois tipos de perturbação:

- ***Perturbação de Dados.*** Modifica os valores da mensagem de acordo com os tipos de dados declarados.

- ***Perturbação de Interação.*** Classifica as mensagens de comunicação em dois tipos: Comunicação por Chamada de Procedimento Remoto e Comunicação de Dados.

Xu *et al.* [XUO05] propuseram um conjunto de operadores de perturbação para XML *Schemas* para criar mensagens inválidas. O XML *Schema* é representado por uma árvore e alguns operadores são definidos para inserir, remover e alterar nós ou sub-árvores no *schema* original. Mensagens XML são derivadas a partir dos XML *Schemas* alterados e enviadas para os *Web Services* em teste, agindo como casos de teste. Para aplicar este método é necessário um XML *Schema*, e em muitas aplicações, incluindo *Web Services*, este *schema* nem sempre está disponível.

Tsai *et al.* [TSA04] propuseram uma técnica que utiliza a teoria de Testes Progressivos de Grupos. *Web* Services com as mesmas funcionalidades são testados em grupo, utilizando um número progressivamente crescente de casos de teste. Contudo, não foi explicitado sobre a existência de alguma implementação para o trabalho.

1.1. Motivação

A tecnologia de *Web Services* ainda é considerada uma tecnologia bastante recente. Muito se tem trabalhado para evoluí-la, no entanto, ainda continua a ser um grande desafio para a comunidade de Tecnologia da Informação [DIL05] devido ao fato de serem aplicações distribuídas e heterogêneas [OFF04], exigindo que requisitos de qualidade como a confiabilidade e a segurança da informação sejam verificados e validados com maior grau de rigidez. A falta de interface com o usuário torna ainda mais difícil a aplicação de um procedimento de teste devido à perda de controlabilidade [DAV02]. Além disso, *Web Services* admitem mudanças no fluxo de dados entre componentes de *software* em tempo de execução, característica conhecida como integração dinâmica. Offutt e Xu [OFF04] ainda comentam que por não estarem disponíveis os detalhes de implementação, o que normalmente só é conhecido pelo fornecedor do serviço, os envolvidos neste procedimento são levados a só utilizarem estratégias funcionais, o que pode esconder boa parte dos problemas trazidos com a integração, gerando resultados de teste com nível de confiabilidade questionável.

Apesar dos trabalhos relacionados apresentarem iniciativas no sentido de introduzir critérios de teste específicos para o contexto de *Web Services*, nenhuma ferramenta foi implementada para facilitar a aplicação destes critérios para dar suporte ao testador em suas atividades. Grande parte das ferramentas consegue executar teste de mensagens SOAP, WSDL e emulação de consumidor e produtor de serviços. Porém, as ferramentas que realizam teste das mensagens SOAP em *Web Services* são centradas na comunicação de chamada a procedimentos remotos, e não incluem comunicação via XML [OFF04].

O teste de perturbação tem se mostrado uma abordagem bastante promissora, entretanto, os operadores de perturbação encontrados na literatura necessitam ser avaliados e refinados.

Verifica-se então que, em relação à evolução das técnicas utilizadas para teste de *Web Services*, ainda há necessidade de novas metodologias e ferramentas capazes de verificar e validar o comportamento e integração destes serviços, assim como facilitar manutenções nestes sistemas de forma rápida e eficiente.

1.2. Objetivos

Este trabalho tem como objetivo explorar a abordagem de perturbação de mensagens no contexto de *Web Services*. Alguns novos operadores são introduzidos para modificar mensagens SOAP diretamente. Para auxiliar a aplicação dos operadores propostos e dos operadores apresentados em [OFF04] uma ferramenta, chamada SMAT-WS, foi descrita e implementada.

A ferramenta permitiu uma avaliação do método de perturbação com a utilização de um estudo de caso formado por um conjunto de *Web Services* reais. As mensagens modificadas foram utilizadas como casos de teste sendo transmitidas aos *Web Services* em teste. As mensagens criadas foram agrupadas por operador de perturbação, derivando diversas métricas para avaliação de sua efetividade.

1.3. Organização do Trabalho

Este trabalho está organizado como segue: o Capítulo 2 apresenta uma visão geral sobre a tecnologia XML e as principais ferramentas de utilização da mesma, como: *parsers*, validadores e transformadores. No Capítulo 3, tem-se uma apresentação da tecnologia de *Web Services* e seus conceitos fundamentais. Em seguida, no Capítulo 4, é apresentada uma visão geral sobre teste de *software*, com enfoque em *Web Services*. No Capítulo 5, são descritos os operadores propostos baseados na teoria de perturbação de dados e a ferramenta implementada para auxiliar a utilização de tais operadores. No Capítulo 6, é descrito o estudo de caso realizado para esta pesquisa. No Capítulo 7 são mostradas as conclusões e trabalhos futuros. O trabalho também contém um apêndice (Apêndice A) com diversas telas da ferramenta implementada assim como um anexo (Anexo I) que apresenta um exemplo de um documento WSDL.

"Desenvolvedores podem construir e testar com base nas especificações, mas os usuários aceitam ou rejeitam com base nas realidades operacionais existentes e reais".
Bernad Boar

2. XML

A *Extensible Markup Language* (XML) é uma linguagem de marcação simples, extensível e flexível para descrição de dados e que pode ser utilizada para definir a sintaxe de outras linguagens [GAB02, WXML]. Ao contrário da HTML, a XML não descreve como os dados serão apresentados, mas estrutura informações de forma a permitir a troca de mensagens entre aplicações.

A linguagem XML foi especificada a partir da *Standard Generalized Markup Language* (SGML) [W3C] e pode ser utilizada para construção de documentos estruturados e auto-descritos regidos por uma série de regras. Na Figura 2.1 é mostrado um esquema da hierarquia das linguagens surgidas a partir de XML.

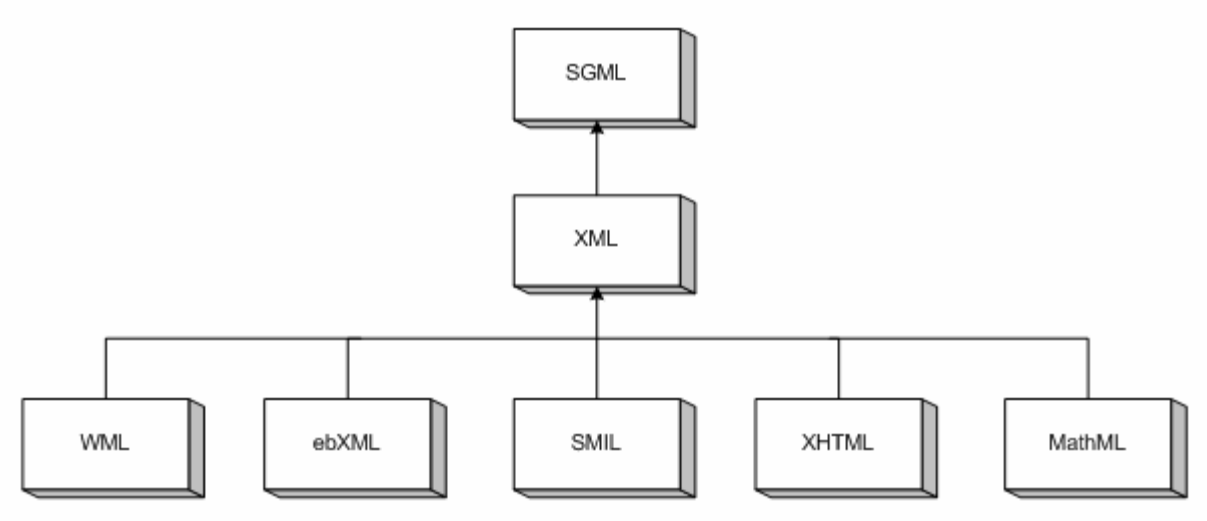

Figura 2.1: Hierarquia entre as linguagens.

Um documento XML é entendido com base em dois conceitos fundamentais:

- ***Marcação.*** Descreve e estrutura o conteúdo do documento XML.
- ***Conteúdo.*** A informação transportada pelo documento XML.

Na Figura 2.2 é mostrado um exemplo de um documento XML bem formado. Um documento XML é dito bem formado quando:

- Contém um ou mais elementos;
- Existe exatamente um elemento, chamado elemento raiz, que não aparece no conteúdo de nenhum outro elemento;
- Para cada elemento não-raiz C no documento, há apenas um elemento P tal que C está contido em P, mas não está contido em nenhum outro elemento contido em P. P é definido como pai de C, e C como filho de P, ou seja, todos os elementos estão aninhados apropriadamente formando uma hierarquia.

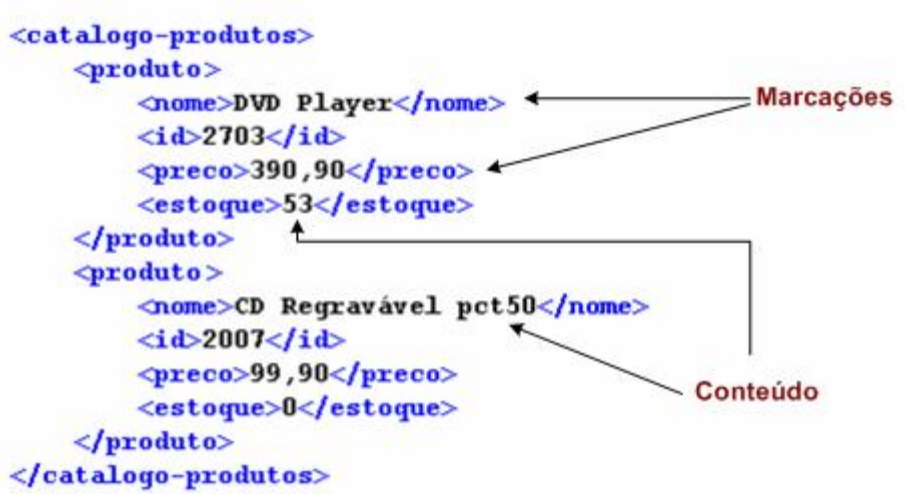

Figura 2.2: Exemplo de documento XML.

Sobre os conceitos de *marcação* e *conteúdo* tem-se toda a flexibilidade e extensibilidade oferecidas pela XML. Estas características permitem que a representação da informação seja facilmente enviada, recebida, ou mesmo, transformada em outros padrões ou formatos de mensagens [CER02]. Com isso, percebe-se, principalmente, um uso considerável e ainda em expansão deste padrão no segmento corporativo, onde a troca de informações entre empresas (*Business to Business* - B2B) com realidades tecnológicas bem distintas é realizada com a utilização de documentos XML, mostrando-se como uma alternativa viável na solução de problemas de integração de sistemas e intercâmbio de dados. Percebe-se então que, com as mais variadas formas de utilização e o potencial de uso da linguagem, a aceitação desta pela comunidade de Tecnologia da Informação está em plena expansão.

Alguns problemas clássicos da área de Tecnologia da Informação foram facilmente solucionados com a introdução da tecnologia XML. Um destes, diz

respeito à integração de sistemas [CER02]. Em grandes corporações, diversas destas soluções conviviam integradas de forma precária e *ad-hoc*, onde as próprias empresas construíam suas soluções que acabavam sendo grandes consumidoras de recursos, tanto financeiros quanto de pessoal. Com a introdução da linguagem XML e sua adoção mundial, o acesso a uma infinidade de padrões e ferramentas fez-se disponível, com duas características adicionais: (1) não seria mais necessário o investimento exorbitante em tecnologias desenvolvidas dentro da própria empresa para soluções de problemas que há tempos deixavam seus líderes preocupados; (2) com a adoção de uma tecnologia com aceitação mundial, as empresas estariam, instantaneamente, aptas a colocarem seus serviços à disposição de qualquer outra empresa ou cliente que utilizasse a mesma estrutura.

2.1. Tecnologias XML

Com a adoção da tecnologia XML pelo mercado, muitas ferramentas surgiram a fim de prover o suporte necessário à utilização desta linguagem. Tais tecnologias são classificadas como segue: tecnologias para validação de XML, p*arsers* XML e transformadores XML.

2.1.1. Validação XML

Com a utilização crescente da linguagem XML, faz-se necessário o uso de ferramentas que nos informem sobre a corretude de um documento em relação a uma determinada gramática. Este processo de verificação é chamado de validação XML e dentre as gramáticas existentes, as mais conhecidas e utilizadas são: *Document Type Definition* (DTD) [GAB02] e XML *Schema Definition* (XSD) [GAB02, XS01]. A seguir são apresentados mais detalhes sobre estas tecnologias de validação de documentos XML.

2.1.1.1. *Document Type Definition*

O primeiro mecanismo de definição de esquemas da linguagem XML foi a *Document Type Definition (*DTD).

A DTD [GAB02] é um arquivo texto contendo um conjunto de regras sobre a estrutura e o conteúdo dos documentos XML. Ela inclui uma lista de elementos que podem aparecer no documento, a ordem destes elementos e seus atributos, informando também se são requeridos ou não. É uma forma bastante simples, rápida e direta de especificação de gramáticas de documentos XML, porém bem rudimentar e incompleta para determinadas situações, pois:

- DTD não gerencia conflitos de *namespaces,* ou seja, dois elementos não podem aparecer na mesma definição;
- Não permite reaproveitamento e nem modularização de definições;
- Não conseguem descrever relacionamentos mais complexos entre documentos e até mesmo elementos;
- Não são modulares e não prevêem reuso nem herança, o que dificulta fortemente as manutenções deste tipo de tecnologia.

Na Figura 2.3 é mostrado um exemplo de uma DTD para validação do documento XML da Figura 2.2. Note que, uma vez de posse da DTD de formação do documento diversas ferramentas, entre elas *IDE´*s (*Integrated Development Environment*) de desenvolvimento, podem fazer uso destas gramáticas a fim de validar os documentos XML em produção.

```
<!ELEMENT estoque (#PCDATA)>

<!ELEMENT preco (#PCDATA)>

<!ELEMENT id (#PCDATA)>

<!ELEMENT nome (#PCDATA)>

<!ELEMENT produto (nome, id, preco, estoque)>
<!ATTLIST produto disponivel CDATA #REQUIRED>

<!ELEMENT catalogo-produtos (produto)*>
```

Figura 2.3: Exemplo de DTD.

2.1.1.2. XML *Schema Definition*

A XML *Schema Definition* [GAB02, XS01], também conhecida como XSD ou apenas XML *Schema*, é uma forma de definição de gramáticas para documentos XML no próprio formato XML. Vem sendo mantida pela *World Wide Web Consortium* (W3C), e foi desenvolvida no sentido de atender as limitações da DTD.

A XSD fornece meios de resolver problemas de *namespaces*, permite formas precisas de definição de tipos de dados simples e complexos, provê meios de herança, modularização e reutilização de definições. Com estas características, a XSD é naturalmente considerada uma evolução natural da DTD, que pouco a pouco, vem sendo substituída.

Na Figura 2.4 é mostrado um exemplo de um XSD para definição da gramática do documento XML da Figura 2.2.

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xs:element name="catalogo-produtos">
        <xs:complextype>
            <xs:sequence>
                <xs:element name="produto" minOccurs="0" maxOccurs="unbounded">
                    <xs:complextype>
                        <xs:sequence>
                            <xs:element name="nome" type="xs:string"/>
                            <xs:element name="id" type="xs:long"/>
                            <xs:element name="preco" type="xs:double"/>
                            <xs:element name="estoque" type="xs:integer"/>
                        </xs:sequence>
                    </xs:complextype>
                </xs:element>
            </xs:sequence>
        </xs:complextype>
    </xs:element>

</xsd:schema>
```

Figura 2.4: Exemplo de XML *Schema Definition*.

2.1.2. *Parsers* XML

Os documentos XML que são utilizados pelas aplicações, em geral, necessitam de um mecanismo que extraia os dados contidos nos documentos XML para atender às suas necessidades de processamento. Os *parsers* XML são responsáveis por este serviço.

Existem diversas implementações de *parsers* XML, sejam estas comerciais ou de caráter aberto, porém, duas formas bastante conhecidas pela comunidade de Tecnologia da Informação são: *Simple API for XML* (SAX) e *Document Object Model* (DOM). Serão mostrados mais detalhes sobre estas duas tecnologias a seguir.

2.1.2.1. *Simple API for XML*

Simple API for XML (SAX) [CER02] é um modelo de *parser* XML baseado em eventos. Seu modo de funcionamento dá-se da seguinte forma: a medida que o *parser* SAX lê o documento XML, simultaneamente, são chamadas as rotinas da aplicação sempre que encontrados elementos XML específicos. Desta maneira, uma aplicação que utiliza um *parser* SAX precisa fornecer implementações de manipuladores de eventos para o *parser*.

Basicamente, os manipuladores de eventos são máquinas de estados finitos que geram novos estados a partir das entradas encontradas no documento XML. Por fazer o processamento em tempo de leitura do documento, *parsers* que utilizam tal abordagem são normalmente mais eficientes e são recomendados em caso de leituras de grandes documentos XML, quando não se tem o interesse de colocá-los completamente em memória. Porém, uma de suas limitações é a inabilidade de ler pontos específicos do documento, se tornando assim uma forma “seqüencial” de leitura de todo o documento em questão.

2.1.2.2. *Document Object Model*

Document Object Model (DOM) [DOM04] é um modelo de *parser* baseado na estrutura de dados de árvores. Seu modo de funcionamento dá-se da seguinte

forma: o *parser* DOM faz a leitura de todo o documento XML em questão, gerando uma árvore de objetos que representa tal documento em memória. Note que, pelo fato de fazer toda a leitura do documento XML, os *parsers* DOM são bem mais custosos em termos de processamento e mais lentos que o modelo SAX, contudo, possuem uma implementação bem mais simplificada, tornando o manuseio do documento em memória bem mais amigável e de fácil manutenção. São recomendados em casos onde desempenho não é um requisito chave para a aplicação e os documentos a serem analisados são de tamanho reduzido.

2.1.3. Transformadores XML

Uma das maiores vantagens da tecnologia XML sobre outras tecnologias é a capacidade de conversão de um documento XML em outro formato qualquer ou mais genérico. Uma das tecnologias mais aceita, utilizada pela comunidade e que permite tal transformação é a tecnologia XSLT (*Extensible Stylesheet Language for Transformations*) [CER02].

2.1.3.1. XSLT

O XSLT [CER02] é um *framework* que permite a conversão de um documento XML em outro formato desejado.

O processador XSLT recebe um *template* XSL, conjunto de regras informando como o documento XML deverá ser tratado para gerar a saída esperada, e o documento XML propriamente dito, contendo as informações necessárias que serão fundidas e transformadas no documento resultante (saída). Seu funcionamento está ilustrado na Figura 2.5:

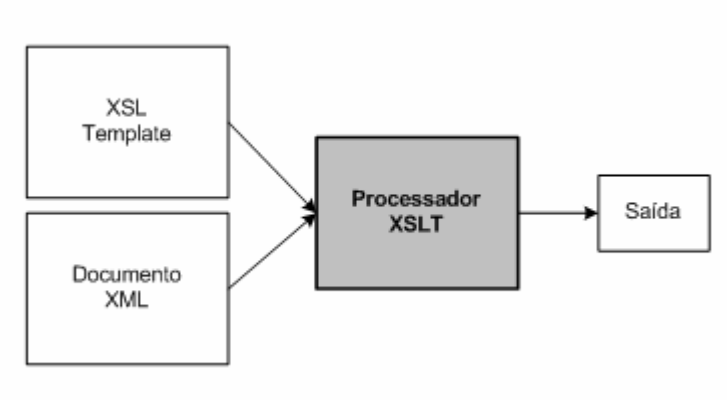

Figura 2.5: Funcionamento de um processador XSLT.

O objeto de saída da Figura 2.5 é o resultado do processamento de fusão entre o *template* e o documento XML. Tal saída pode tomar qualquer forma desejada, bastando que se tenha informado os *templates* adequados. Um formato bastante utilizado atualmente é a transformação de XML em documentos HTML. Outros exemplos de documentos (saídas) produzidos por um processador XSLT incluem: mensagem SOAP, documento XML, documento texto, entre outros.

2.2. Considerações Finais

Este capítulo apresentou a linguagem XML e as principais tecnologias envolvidas. Foram apresentadas as formas mais utilizadas de gramáticas de validação (DTD e XML *Schema*), assim como os principais *parsers*, responsáveis pelo mecanismo de extração de dados de documentos XML, e transformadores, capazes de converter um documento XML em um outro formato.

A integração entre sistemas exige uma padronização da linguagem a ser adotada na troca de informações. Esta padronização vem acontecendo e cada vez mais a linguagem XML vem sendo utilizada nos mais diversos cenários. A tecnologia de *Web Services* tem como um de seus principais objetivos a integração entre sistemas, adotando a linguagem XML como base para troca de informações entre os serviços e para definição de protocolos utilizados com esta tecnologia, como: SOAP, WSDL e UDDI. No capítulo seguinte serão mostrados os detalhes e todas as especificações envolvidas com a tecnologia de *Web Services*.

"A interoperabilidade não é apenas mais uma questão de qualidade, mas também de poder".

"A *World Wide Web* está sendo cada vez mais usada para comunicação entre aplicações. As interfaces utilizadas nestas comunicações são chamadas de *Web Services"* [WSA04].

3. *Web Services*

De acordo com a *World Wide Web Consortium (W3C), Web Service* é um sistema de *software* projetado para permitir interações entre computadores dentro de uma rede [WSA04].

Esta definição se torna bastante abrangente quando não se determina o escopo de duas das suas principais implicações: sistema de *software* e interação. Entende-se por sistema de *software*, neste contexto, uma infra-estrutura formada por um conjunto integrado de protocolos, especificações e linguagens. Este conjunto é capaz de fornecer um ambiente homogêneo para comunicação (interação) entre elementos heterogêneos daquele sistema.

Dentro dos próximos anos, a tecnologia de *Web Services* promete mudar fortemente o cenário da computação distribuída como é conhecida atualmente [BLO02b] e abrir novas possibilidades e oportunidades de negócio para as mais diversas empresas. Construída sobre um modelo comum de comunicação, esta tecnologia baseia-se em protocolos padrões e bem conhecidos pela comunidade da Internet [WSA04].

A utilização de *Web Services* permite que aplicações sejam integradas mais rapidamente e a custos mais baixos em relação a outras tecnologias. Estas características são ideais para sua adoção na comunicação entre empresas distintas, assim como dentro do perímetro de uma mesma empresa para integração de seus sistemas [KRE01].

Web Services, então, tratam de interoperabilidade, contudo, não só interoperabilidade entre tecnologias, como também entre protocolos, plataformas e sistemas. As implicações dessa afirmação são significativas, contudo se tornam ainda mais claras quando se sabe que é indiferente para a comunicação:

- O tipo de aplicação ou sobre que sistema operacional ela está sendo executada;
- De que lugar no mundo os sistemas estão enviando as mensagens;
- Linguagem de programação que a aplicação foi escrita;
- *Hardware* dos sistemas;
- Sistema operacional em execução.

Resumindo, toda aplicação de *software* é, então, potencialmente capaz de comunicar-se com qualquer outra aplicação no mundo, superando todos os limites impostos por barreiras geográficas, sistema operacional, linguagem de programação, protocolos, *hardware*, etc [WSDL01].

3.1. Histórico

A interoperabilidade é considerada um quesito de qualidade que vem crescendo em importância para a Tecnologia da Informação. De forma geral, pode ser definida como sendo a habilidade de produtos, sistemas ou processos de negócio de trabalhar em conjunto para solução de uma tarefa comum. Em um conceito mais específico de *software* a *Institute of Electrical and Electronics Engineers* (IEEE) define interoperabilidade como sendo a habilidade de dois ou mais sistemas ou componentes em trocar informações e utilizar a informação que foi transferida [IEEE90].

Anterior a tecnologia de *Web Services*, diversas outras soluções técnicas foram propostas para resolver questões de interoperabilidade entre sistemas. Dentre elas destacam-se CORBA [ORG06], RMI [SUN06] e DCOM [HOR06], exemplos existentes até hoje convivendo de forma paralela no mercado.

CORBA é um acrônimo para *Common Object Request Broker Architecture*. Foi proposta e vem sendo mantida pela *Object Managment Group* (OMG) desde 1991. CORBA é a especificação de uma arquitetura padrão para sistemas de objetos distribuídos, desenvolvida para prover interoperabilidade entre tais objetos via rede. É uma arquitetura madura, bastante utilizada para aplicações de missão crítica e baseada no protocolo chamado *Internet Inter-ORB Protocol* (IIOP) para

comunicação entre os objetos distribuídos. Contudo, por não ser baseada em tecnologias e protocolos amplamente conhecidos pela comunidade da Internet, um dos seus principais desafios é torná-la acessível via *web* [MUR02].

Remote Method Invocation (RMI) é uma tecnologia desenvolvida pela Sun em 1997 e que permite ao desenvolvedor criar aplicações distribuídas na linguagem Java. Uma de suas principais características é permitir a chamada de um método implementado e localizado em um objeto remoto. Mais importante, a chamada de um método em um objeto remoto tem a mesma sintaxe que uma chamada de um método em um objeto local. RMI é baseada no protocolo chamado *Java Remote Method Protocol* (JRMP) e considerada a versão Java do que é conhecido pela comunidade como *Remote Procedure Call* (RPC). Uma de suas principais desvantagens é sua interoperabilidade, restrita apenas a objetos Java.

Distributed Component Object Model (DCOM) é uma extensão da *Component Object Model* (COM). Desenvolvida pela Microsoft em 1997, esta tecnologia também permite a comunicação entre objetos em diferentes computadores através do protocolo chamado *Object Remote Procedure Call* (ORPC), estejam eles na mesma rede ou até na Internet. Algumas de suas características envolvem o fato de estar fortemente ligado à plataforma Windows e objetos DCOM poderem ser escritos e invocados em uma variedade de linguagens desde que suportem serviços COM.

Não faz parte do escopo deste trabalho descrever em detalhes as tecnologias envolvidas na produção de Sistemas Distribuídos, no entanto, em resumo, as arquiteturas das tecnologias CORBA, RMI e DCOM possuem mecanismos transparentes de chamada a objetos remotos. Apesar de implementarem tais mecanismos de formas diferentes, as idéias utilizadas por cada uma delas são bastante similares.

3.2. Arquitetura

3.2.1. Arquitetura Orientada a Serviços

Aplicações distribuídas modernas são construídas de acordo com a Arquitetura Orientada a Serviços (*Service Oriented Architecture – SOA*) [HEC04] cuja representação é mostrada na Figura 3.1.

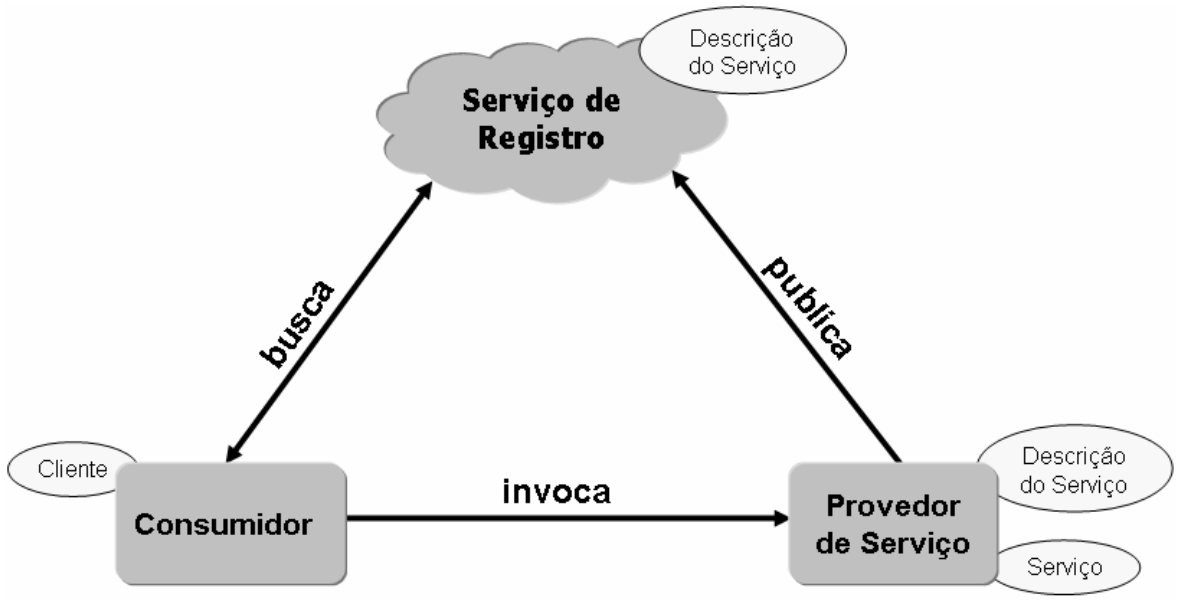

Figura 3.1: Arquitetura Orientada a Serviços.

3.2.1.1. Papéis

A Arquitetura Orientada a Serviços envolve três papeis fundamentais, a saber:

- **Provedor de Serviço.** Hospeda o serviço e provê acesso aos consumidores (clientes). Um serviço pode ser definido como uma funcionalidade independente que pode ser executada remotamente por outros componentes de *software*. O provedor de serviço também é responsável por publicar a descrição do seu serviço no Serviço de Registro.
- **Consumidores.** São os componentes de *software* que necessitam que alguma funcionalidade específica seja atendida. Alguns exemplos de consumidores envolvem navegadores *web*, um programa, etc.

- **Serviço de Registro.** Repositório contendo as descrições dos serviços publicados pelos provedores de serviço. Além de guardar informações sobre os serviços, o registro também possui a responsabilidade de fornecer métodos de busca que serão utilizadas pelos consumidores. Logo, percebe-se que acoplamentos estáticos entre provedores e consumidores de serviço não necessitam da utilização de um Serviço de Registro.

3.2.1.2. Operações

A Arquitetura Orientada a Serviços define três operações, a saber:

- **Publicação.** Para ser acessado, um serviço deve ser publicado. Uma vez publicado o serviço poderá ser encontrado e em seguida utilizado pelos consumidores.
- **Busca.** O consumidor recebe uma descrição de serviço diretamente ou pode fazer uma busca no registro pelo tipo de serviço requerido. Percebe-se que o serviço de registro pode ser utilizado em dois momentos distintos: na fase de construção do *software*, onde a interface do serviço pode ser encontrada para ser integrada no componente em construção, ou em tempo de execução onde o próprio componente de *software* é responsável por buscar e efetuar as chamadas necessárias para a execução da funcionalidade requerida.
- **Chamada.** Uma vez de posse dos dados do serviço, o consumidor poderá então invocar ou chamar a funcionalidade desejada. Neste momento ocorre uma interação entre provedor e consumidor de serviço onde é pedida a execução de algum processamento e é enviada, posteriormente, a resposta obtida de tal computação.

3.2.1.3. Artefatos

São definidos dois artefatos que fazem parte da Arquitetura Orientada a Serviços, a saber:

- **Serviço.** O serviço é a implementação de um produto de *software* capaz de ser distribuído através de uma rede de comunicação. Ele existe para fornecer algum processamento ou computação para possíveis e interessados consumidores.
- **Descrição do Serviço.** Contém os detalhes da interface e implementação do serviço, incluindo seus tipos de dados, operações, localização na rede, entre outros. A descrição do serviço é publicada em um serviço de registro para que seja buscada por consumidores de serviço a fim de fornecer as informações básicas para que se tenha interoperabilidade entre sistemas.

3.2.2. Integração

A Arquitetura Orientada a Serviços envolve os três papéis definidos anteriormente: Consumidor, Provedor de Serviço e Serviço de Registro como mostrado na Figura 3.1. O Provedor de Serviço expõe alguma funcionalidade de *software* como um serviço para seus clientes. Como exemplo, uma destas funcionalidades pode ser um serviço de busca de produtos em um catálogo de vendas de uma determinada empresa. Para que a funcionalidade esteja visível para todos os clientes, o Provedor de Serviço publica a descrição do serviço, uma vez que ele e o Consumidor não se conhecem a priori. Esta publicação é feita em uma entidade especial chamada Serviço de Registro. Uma vez registrado, o serviço poderá ser buscado através de pesquisas feitas pelo Consumidor. Uma vez que o que encontre algum serviço que lhe interessa no Serviço de Registro, o Consumidor requisita as informações do serviço e a partir de então estará apto a invocar as funcionalidades do Provedor de Serviço. Com isso fecha-se o ciclo *Publica-Busca-Invoca* que caracteriza a Arquitetura Orientada a Serviços. Uma outra característica desta arquitetura é o dinamismo devido ao baixo acoplamento entre os serviços e ao fato de que os próprios clientes podem invocar tais serviços em tempo de execução.

O mecanismo para alcançar esta visão de uma arquitetura dinâmica e aberta depende de duas suposições [HEC04]:

1. Os serviços fornecidos estão funcionalmente corretos em relação aos seus requisitos e sua descrição.
2. Descrições que se igualam as descrições dos requisitos são suficientes para estabelecer interoperabilidade entre consumidor e provedor de serviços.

A primeira suposição é trivial e necessária para todos os tipos de sistema e pode ser garantida através de testes de unidade. A segunda suposição deriva do fato que o baixo acoplamento de sistemas distribuídos não pode ser testado quanto a sua integração. Desta forma, as especificações têm que conter informações suficientes sobre o serviço, ou seja, uma vez que o consumidor encontra um serviço apropriado, provedor e consumidor podem trabalhar de forma conjunta a partir de então [HEC04].

3.2.3. Arquitetura de um *Web Service*

A arquitetura de *Web Service* forma um conjunto de funcionalidades colocadas à disposição de aplicações remotas e serviços através da Internet. É considerada uma implementação da Arquitetura Orientada a Serviços e é construída sobre protocolos de comunicação que fazem a base da Internet, como: HTTP, FTP, SMTP.

As principais tecnologias envolvidas com a utilização de *Web Services* são: XML, *Simple Object Access Protocol* (SOAP - protocolo de mensagem), *Web Service Description Language* (WSDL - interface de definição dos serviços) e *Universal Description, Discovery and Integration* (UDDI - registro de publicação de *Web Services*). Todas estas tecnologias compõem a estrutura básica de um *Web Service* para fornecer um serviço de interoperabilidade entre sistemas.

De forma geral, o documento XML representa a informação sendo trocada pelos serviços. SOAP fornece um mecanismo de empacotar e rotear o documento XML através da rede. A WSDL define um formato para descrever o serviço e sua interface externa. Finalmente, o registro UDDI permite que *Web Services* sejam registrados de uma maneira uniforme, fornecendo aos clientes um serviço de localização de serviços. A seguir serão descritas em maiores detalhes cada uma das tecnologias

envolvidas com *Web Services*. A Figura 3.2 mostra como cada parte se posiciona dentro da interação do usuário com um *Web Service* qualquer.

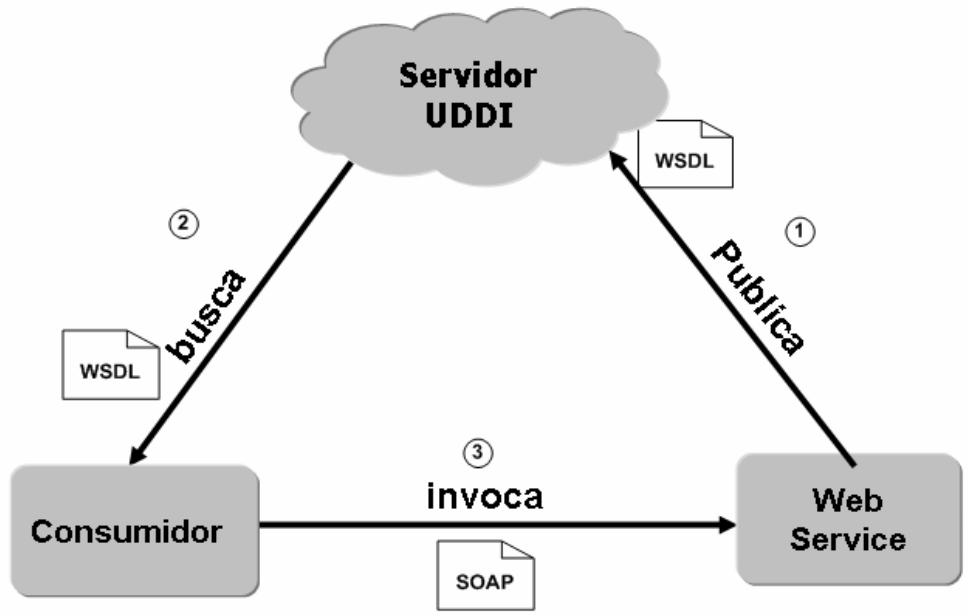

Figura 3.2: Diagrama de interação com um *Web Service*.

Na Figura 3.2 é mostrado como se dá a interação completa entre um *Web Service* e um consumidor qualquer. No momento 1 o *Web Service* publica o seu serviço no servidor UDDI. Em seguida (momento 2) o consumidor busca no servidor UDDI algum serviço que se adeque aos seus requisitos. Em seguida (momento3), de posse do WSDL do serviço desejado, o consumidor envia uma requisição SOAP para o *Web Service* em questão requisitando a execução daquele serviço, que retornará uma outra mensagem SOAP contendo os resultados do processamento desejado, encerrando-se assim o fluxo de execução. Considere que um consumidor possui todas as informações necessárias para requisitar a execução de algum serviço. Percebe-se que, neste caso, a busca que poderia ser dirigida ao servidor UDDI à procura de tal serviço se faz desnecessária.

Na Figura 3.3 são mostradas as opções de tecnologias utilizadas em conjunto com *Web Services* e a camada as quais estão inseridas.

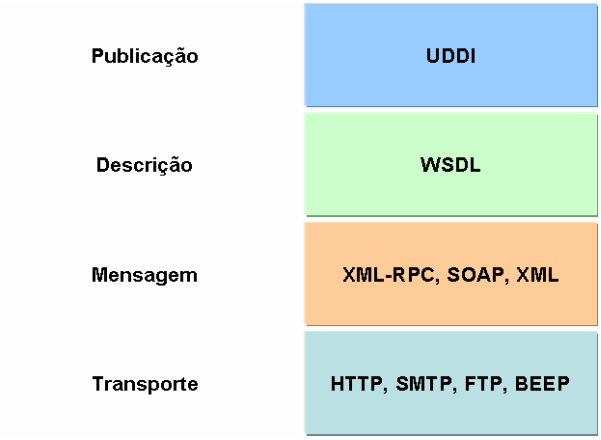

Figura 3.3: Pilha de protocolos de *Web Service*.

Nas próximas seções serão descritos em mais detalhes os protocolos e especificações integrantes da tecnologia de *Web Services*.

3.3. SOAP

SOAP [CER02, SOAP03] é um protocolo baseado em XML para troca de informações entre computadores. Este protocolo define a forma em que os dados serão codificados assim como as regras de empacotamento para as informações envolvidas na comunicação. Apesar de poder ser utilizado com uma variedade de sistemas de mensagem e roteamento via uma variedade de protocolos de transporte, o foco inicial do SOAP são as Chamadas de Procedimentos Remotos (*Remote Procedure Calls* - RPC) sobre HTTP.

O protocolo SOAP pode ser utilizado de duas formas distintas:

- ***Síncrona*.** A chamada SOAP interrompe o fluxo de execução do cliente até a chegada de uma resposta, também no formato SOAP, enviada pelo serviço requisitado.
- ***Assíncrona*.** As mensagens que trafegam entre os serviços são armazenadas em um servidor de mensagens podendo ser consumidas a qualquer momento. Cabe ressaltar que nesta forma de utilização os *Web Services* não necessitam interromper os fluxos de execução, podendo dar continuidade ao processamento normalmente.

A especificação SOAP endereça três áreas principais:

- ***Especificação de Envelope*.** Define regras para encapsulamento de dados sendo transferidos entre computadores. Isto inclui dados específicos de aplicação como: nome do método a ser invocado, parâmetros ou valores de retorno. Também inclui informação de quem deveria processar o conteúdo do envelope, e no caso de falhas, como codificar mensagens de defeito.
- ***Regras de codificação de dados*.** Para trocar informações, computadores precisam padronizar regras para codificação de tipos de dados específicos. Muitos destes padrões estão contemplados na especificação do XML *Schema*.
- ***Convenções RPC*.** SOAP pode ser usado em uma variedade de sistemas de comunicação, incluindo comunicação unidirecional ou mesmo bidirecional. Para comunicações bidirecionais, SOAP também define, como uma das formas de comunicação, uma convenção simples para representar chamadas de procedimento remoto e suas respostas.

Um exemplo de uma mensagem de requisição em SOAP e uma resposta estão representados na Figura 3.4 e na Figura 3.5, respectivamente. Note que as três áreas endereçadas nas mensagens SOAP estão mapeadas através das marcações *Envelope*, *encodingStyle* e *Body.*

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsi=http://www.w3.org/2001/XMLSCHEMA-instance
  xmlns:xsd=http://www.w3.org/2001/XMLSCHEMA
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:GetInfoProduto xmlns:ns1="MinhaEmpresa-URI">
      <produtoId xsi:type="xsd:long">123456</produtoId>
    </ns1:GetInfoProduto>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 3.4: Exemplo de mensagem SOAP de requisição.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsi=http:/b/www.w3.org/2001/XMLSCHEMA-instance
  xmlns:xsd=http://www.w3.org/2001/XMLSCHEMA
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:GetInfoProdutoResposta xmlns:ns1="MinhaEmpresa-URI">
      <return>
        <Descricao>DVD Player</Descricao>
        <Preco>349,90</Preco>
      </return>
    </ns1:GetInfoProdutoResposta>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 3.5: Exemplo de mensagem SOAP de resposta para mensagem da Figura 3.4.

3.3.1. Estrutura

Esta seção descreve como está estruturada uma mensagem SOAP. Uma mensagem SOAP é definida como sendo uma mensagem XML formada por:

- Um elemento *Envelope* como nó raiz;
- Um elemento *Header* [opcional];
- Um elemento *Body* [obrigatório];
- Um elemento *Fault* utilizado apenas em casos de defeito [opcional].

A Figura 3.6 mostra uma representação didática da estrutura de uma mensagem SOAP.

```
    <Envelope>
      <Header>
      </Header>
      <Body>
            <Fault>
            </Fault>
      </Body>
    </Envelope>
```

Figura 3.6: Estrutura de uma mensagem SOAP.

Os elementos que integram uma mensagem SOAP são detalhados a seguir.

Envelope

Assim como a linguagem XML define um nó raiz para todas as mensagens, uma mensagem SOAP possui um elemento raiz chamado *Envelope.* Tal marcação define qual a versão utilizada, qual o estilo de codificação a ser utiliza, quais os *namespaces* utilizados na mensagem, entre outras funções. Um exemplo deste elemento é mostrado na Figura 3.7.

```
<SOAP-ENV:Envelope
Xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
Xmlns:xsi=http://www.w3.org/2001/XMLSCHEMA-instance
Xmlns:xsd=http://www.w3.org/2001/XMLSCHEMA
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

Figura 3.7: Exemplo do campo *Envelope* em uma mensagem SOAP.

Header

A marcação opcional *Header* oferece um *framework* para especificação adicional de requisitos no nível de aplicação. Por exemplo, este elemento pode ser utilizado para especificar uma assinatura digital para serviços protegidos por senha. Um exemplo deste elemento é mostrado na Figura 3.8.

```
<SOAP-ENV:Header>
  <ns1:PaymentAccount xmlns:ns1="urn:ecerami"
    SOAP-ENV:mustUnderstand="true">
    orsenigo473
  </ns1:PaymentAccount>
</SOAP-ENV:Header>
```

Figura 3.8: Exemplo do campo *Header* em uma mensagem SOAP.

Body

O elemento *Body* é um elemento obrigatório para toda mensagem SOAP. Este elemento contém as informações em tráfego entre os serviços. Também é utilizado para transportar o resultado de um processamento, assim como, um relato de

defeito ocorrido no serviço pedido. Um exemplo deste elemento é mostrado na Figura 3.9.

```
<SOAP-ENV:Envelope
  Xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsi=http://www.w3.org/2001/XMLSCHEMA-instance
  xmlns:xsd=http://www.w3.org/2001/XMLSCHEMA
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:GetInfoProduto xmlns:ns1="MinhaEmpresa-URI">
      <produtoId xsi:type="xsd:long">123456</produtoId>
    </ns1:GetInfoProduto>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 3.9: Exemplo de campo *Body* em uma mensagem SOAP.

Fault

Em caso da ocorrência de algum problema no *Web Service,* o servidor retornará uma mensagem de defeito com o elemento *Fault,* contido no elemento *Body,* apresentando detalhes sobre o defeito ocorrido. Um exemplo deste elemento é mostrado na Figura 3.10.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode xsi:type="xsd:string">
        SOAP-ENV:Client
      </faultcode>
      <faultstring xsi:type="xsd:string">
        Failed to locate method (ValidateCreditCard) in class
        (examplesCreditCard) at /usr/local/ActivePerl-5.6/lib/
        site_perl/5.6.0/SOAP/Lite.pm line 1555.
      </faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 3.10: Exemplo de campo *Fault* em uma mensagem SOAP.

A Tabela 3.1 apresenta os sub-elementos do elemento *Fault* e suas funcionalidades.

Tabela 3.1: Sub-elementos do elemento *Fault*.

Elemento	Descrição
Code	Código para informar a classe do defeito encontrada.
Reason	Explicação do defeito encontrado.
Node	Provê informação sobre qual elemento na mensagem SOAP causou o problema.
Role	Especifica o papel que o nó estava operando quando o problema ocorreu. Os papéis podem ser: *next, none, ultimateReceiver.*
Detail	Elemento utilizado para carregar mensagens de defeito específicas da aplicação

A Tabela 3.2 apresenta os possíveis defeitos embutidos nas mensagens de defeito retornadas pelo servidor e suas descrições.

Tabela 3.2: Códigos de defeitos em uma mensagem SOAP.

Elemento	Descrição
SOAP-ENV: VersionMismatch	Indica que o elemento *Envelope* da mensagem inclui um *namespace* inválido, significando um problema de versão
SOAP-ENV: MustUnderstand	Indica que o receptor não consegue processar um elemento *Header* com um atributo *MustUnderstand* setado para *true*
SOAP-ENV: Client	Indica que a requisição do cliente contém defeito
SOAP-ENV: Server	Indica que o servidor não consegue processar a requisição do cliente

3.4. WSDL

Assim como protocolos de comunicação e formatos de mensagens são padronizados na comunidade *web*, também se torna importante a capacidade de descrever a própria comunicação de uma maneira estruturada e padronizada. WSDL possui a responsabilidade de definir tal comunicação [WSDL01].

A *Web Service Description Language* (WSDL) é uma linguagem para especificar a interface de um *Web Service* através de uma gramática comum em XML, representando um contrato entre o provedor do serviço e o cliente. A informação contida na descrição de um *Web Service* inclui endereço de rede, protocolo e um conjunto de funcionalidades/operações disponíveis [WSDL01].

Com a utilização da WSDL um cliente pode saber o endereço de um *Web Service* e invocar qualquer uma das suas funções públicas disponíveis. Assim, esta linguagem é uma base para a arquitetura desta linguagem, provendo uma forma

uniforme para descrever tais serviços, como também uma plataforma para sua integração.

Um documento WSDL utiliza os seguintes elementos na definição de serviços de rede:

- ***Types***. Define os tipos de dados utilizando alguma gramática de validação, como XSD (XML *Schema Definition*).
- ***Message***. Uma definição abstrata dos dados que estão sendo comunicados. Uma mensagem consiste de uma ou mais partes lógicas associadas com os tipos definidos. Quando o modelo SOAP RPC é utilizado, cada parte representa um parâmetro de um método.
- ***Operation***. Uma descrição abstrata de um serviço fornecido pelo *Web Service*.
- ***Port Type***. Um conjunto abstrato de operações. Cada elemento define uma operação e as mensagens de entrada e saída associadas a operação. Quando o modelo SOAP RPC é utilizado, cada operação representa um método.
- ***Binding***. Um protocolo concreto e uma especificação de formato de dados para uma porta específica. Mapeia as operações e as mensagens definidas em uma porta para um protocolo e um formato de dados. Por exemplo, um elemento *binding* pode mapear uma porta para uma interface SOAP RPC utilizando o protocolo HTTP e o sistema de codificação de dados utilizado pelo SOAP.
- ***Port***. Um serviço definido como uma combinação de um *binding* e um endereço de rede.
- ***Service***. Coleção de serviços relacionados. Mapeia um elemento *binding* com a localização (URL) do *Web Service*.

O Anexo I mostra um exemplo de documento WSDL extraído de [WSDL01].

3.5. UDDI

Após a construção e configuração de um *Web Service*, é necessário um meio de publicar, buscar e utilizar tais serviços. Com esse intuito foi criada a especificação UDDI.

A *Universal Description, Discovery and Integration* (UDDI) [CER02] é uma especificação técnica para descrever, localizar e integrar *Web Services*. É também parte fundamental da especificação permitir que organizações publiquem e encontrem serviços. Em suma, é uma especificação para construção de um diretório distribuído contendo informações de empresas e serviços oferecidos por estas empresas.

As informações armazenadas em um servidor UDDI se enquadram em uma das três categorias:

- ***White Pages***. Incluem informações gerais sobre as empresas, como nome, descrição, informações de contato, endereço e telefones. Também pode incluir identificadores únicos de empresas como o Dun & Bradstreet D-U-N-S [DUNS05].
- ***Yellow Pages***. Incluem dados de classificação tanto para empresa quanto para o serviço oferecido. Por exemplo, estes dados podem incluir produto, códigos geográficos, entre outros.
- ***Green Pages***. Contém informações técnicas sobre o *Web Service*. Geralmente, este aponta para uma especificação externa e um endereço para invocar o *Web Service*.

Os registros UDDI funcionam como *Web Services*. Todas as API's (*Application Programming Interface*) nesta especificação também são definidas em XML, empacotadas em envelopes SOAP e enviadas através do protocolo HTTP. Além disso, as requisições de cliente que exigem modificação de dados precisam ser protegidas e autenticadas.

3.5.1. Tipos de Descoberta

Há duas visões de como a descoberta de um *Web Service* pode ocorrer usando um registro UDDI. A primeira, descoberta em tempo de projeto, envolve intervenção humana e alguma programação. A segunda, descoberta em tempo de execução, é mais automática e exige considerável esforço de programação.

3.5.1.1. Descoberta em tempo de projeto

Se uma empresa considera usar um *Web Service*, é preciso que ela decida qual deles será. Uma maneira de fazer isso é por um processo de descoberta manual. O procedimento básico é composto pelas seguintes atividades:

- Os analistas da empresa pesquisam o UDDI para encontrar os *Web Services* candidatos;
- Um programador pesquisa o registro à procura de um *bindingTemplate* com o qual deseja se conectar;
- O programador escreve um programa cliente para o *Web Service* de forma a trocar informações de um modo estático, onde as informações de tal *Web Service* não poderão ser modificadas até que outra versão do *software* seja lançada.

3.5.1.2. Descoberta em tempo de execução

No futuro, o procedimento manual será eliminado e a descoberta em tempo de execução será mais comum [NEW02]. Esta solução, mais avançada e futurista, envolve a descoberta e a interconexão dos clientes aos *Web Services*. As etapas fundamentais deste processo estão relacionadas abaixo:

- Os analistas da empresa especificam que tipo de *Web Services* desejam;
- Um programa de descoberta pesquisa o registro UDDI à procura de um *bindingTemplate* com o qual o cliente possa se conectar em função dos protocolos que ele já suporta;

- O programa de descoberta verifica se o programa cliente pode enviar requisições ao serviço;
- O programa cliente acessa o *Web Service*, transfere o documento e recebe a resposta.

Um dos maiores problemas neste caso de descoberta de serviços é a confiança. Em um mundo de *hackers*, é difícil ver como as empresas poderiam confiar em qualquer *Web Service* a ponto de permitir interconexão automática, porém esta é uma questão que está sendo bastante discutida e não faz parte do escopo deste trabalho.

3.6. Considerações Finais

De acordo com a W3C [WSDL01], *Web Service* é um *software* desenvolvido para permitir interoperabilidade entre tecnologias, assim como, protocolos, plataformas e sistemas operacionais. Conseqüentemente, todos os *softwares* são potencialmente capazes de se comunicar com outros, superando barreiras geográficas, sistema operacional, linguagens de programação e protocolos.

Construída sobre os conceitos da Arquitetura Orientada a Serviços (*Service Oriented Architecture* – SOA), a tecnologia de *Web Services* tem o potencial de reduzir a complexidade e custos de integração de projetos de *software*. É composta por um protocolo de comunicação (SOAP) responsável por transmitir as mensagens entre os sistemas, uma linguagem de descrição de serviços (WSDL) e um registro distribuído de informações sobre *Web Services* (UDDI). Estas três tecnologias formam a base para utilização e construção de *Web Services*.

Verifica-se que *Web Services* oferecem novos cenários para utilização de novas técnicas e ferramentas de teste de *software*, uma vez que possuem um conjunto de características diferenciadas, como heterogeneidade, distribuição dos serviços e ausência de interface com usuário. Estas características tornam complexa a aplicação de procedimentos de teste nestes serviços necessitando assim de um conjunto de métodos que forneçam maior eficiência e suporte ao testador em suas tarefas. No Capítulo 4 serão apresentadas algumas abordagens de teste de *Web*

Services que além de reduzirem estes problemas, formam a base para este trabalho.

"O desenvolvimento de sistemas de *software* envolve uma série de atividades de produção que as oportunidades para injetar a falibilidade humana são enormes. Por causa da inabilidade humana de realizar e de se comunicar com perfeição, o desenvolvimento é acompanhado por uma atividade de controle de qualidade" [DEU79].

4. Teste de *Software*

O processo de teste de *software* há muito tempo desafia os profissionais da área de Engenharia de *Software*. Normalmente, esta é uma atividade que consome grandes recursos durante o ciclo de desenvolvimento de uma aplicação e que demanda um grupo de especialistas, seja no planejamento das atividades quanto na sua execução, uma vez que exige um perfil diferenciado dos envolvidos nestas tarefas. Contudo, os testes são de extrema importância em qualquer linha de desenvolvimento de *software* atual, configurando-se em um grande fator de qualidade do processo como também do produto em construção [ROC01].

A disciplina de teste é considerada uma atividade de verificação e validação. Consiste na análise dinâmica da aplicação, ou seja, na execução do produto de *software* com objetivo de verificar a presença de defeitos e aumentar a confiança de que o produto esteja correto [ROC01]. Contudo, o procedimento de teste não envolve esforços associados à depuração de *software* ou mesmo à remoção/reparo destes defeitos. Assim, os testes e a depuração se configuram em duas atividades completamente distintas, apesar de, em certos casos, serem realizadas pelas mesmas pessoas em um determinado time de desenvolvimento.

Alguns, erroneamente, confundem a atividade de teste com o processo de garantia de qualidade. Contudo, a atividade de teste é necessária, porém insuficiente para garantia de qualidade de um produto. Nenhuma quantidade de testes irá aumentar a qualidade ou garantir a corretude de um *software*, mas apenas ajudar a identificar defeitos que serão removidos pelos desenvolvedores posteriormente.

Normalmente, o procedimento de testes envolve quatro etapas distintas que devem ser inseridas ao longo de todo o processo de desenvolvimento de *software*. São elas [PRES02]:

- ***Planejamento***. Identifica e descreve os testes que serão implementados e executados. Isto é conseguido através da geração de planos de testes contendo os requisitos a serem testados e as estratégias de testes a serem utilizadas.
- ***Projeto de Casos de Teste***. Identifica, descreve e gera o modelo de testes e seus artefatos, como: procedimento de testes e casos de teste.
- ***Execução***. Efetua a execução dos mais diversos testes descritos no plano de testes formulado em etapas anteriores. O resultado desta fase gera informações importantes para a próxima etapa de avaliação.
- ***Avaliação de Resultados***. Medir, avaliar e acompanhar o resultado da execução dos testes a fim de fornecer informações suficientes a respeito do próprio processo de testes. Tais informações poderão ser utilizadas para avaliações de eficiência do processo, cobertura dos testes, custos, qualidade do processo entre diversas outras variáveis.

4.1. Princípios de Testes

Como dito anteriormente, a fase de teste é de extrema importância para a maioria das metodologias de desenvolvimento de *software*, já que é um grande indicador de qualidade tanto do processo, quanto do próprio produto em construção. Porém, é interessante também que tal atividade permeie todas as fases do processo, garantindo que cada artefato produzido seja testado individualmente e de forma integrada durante as diversas evoluções do *software*, evitando a existência de uma fase única e de tamanho considerável de teste, onde sua qualidade e eficácia são sempre questionadas.

O processo de teste é um procedimento dispendioso dentro de qualquer realidade de produção de *software*, podendo em alguns casos atingir 30% do custo total de desenvolvimento, sendo assim, não deve ser executado sem passar por uma engenharia (Engenharia de Testes) bem definida e planejada previamente [PRES02].

Além disso, apesar de todo o custo envolvido com o procedimento de testes de *software*, estes não podem garantir a ausência de defeitos, mas apenas mostrar que defeitos estão presentes em um produto. Verifica-se então que não é possível testar um sistema por completo, ou seja, não é possível garantir que foram cobertos, testados ou mesmo exercitados todos os caminhos ou estados de um programa, por mais que este tenha sido submetido a um rigoroso procedimento de teste.

Infelizmente, um teste exaustivo além de impraticável apresenta certos problemas logísticos. Mesmo para pequenos programas, o número de possíveis caminhos lógicos pode ser muito grande. Considere, por exemplo, um programa com aproximadamente 100 linhas de código composto por dois ciclos aninhados e quatro construções se-então-senão. Em um programa com estas características há aproximadamente 10^{14} caminhos possíveis a serem executados [PRES02]. Com um número tão elevado de caminhos é impraticável percorrer todas as possibilidades de um *software* de tamanho considerável.

Para realizar testes em um *software* no intuito de revelar seus defeitos, escolher os melhores caminhos ou casos de teste com maior probabilidade de encontrar defeitos, diversos critérios de teste foram estudados e propostos.

Um critério de teste é um predicado que deve ser satisfeito para que a atividade de teste seja considerada satisfatória ou mesmo encerrada. Dito de outra forma, o critério de teste informa quando é possível considerar que o programa foi testado suficientemente [RAP85]. Critérios de teste auxiliam nas etapas de seleção e avaliação de um conjunto de casos de teste, de modo que sejam escolhidos aqueles que tenham maior probabilidade de revelar defeitos ainda não revelados. O aumento no nível de confiança na corretude do programa também é um fator importante com o uso de critérios de teste. Desta forma, considera-se que os critérios de teste estabelecem formas de medições de cobertura de *software*, onde a cobertura de *software* é definida como uma métrica que informa a proporção de um programa exercitada por um determinado conjunto de casos de teste em relação a um determinado critério de teste.

Os critérios de teste foram estabelecidos considerando diferentes perspectivas e podem ser classificados da seguinte forma:

- ***Critérios Funcionais*.** Centrado nos requisitos funcionais do sistema/*software*. Não há preocupação com detalhes de implementação. Alguns exemplos de critérios funcionais são: Particionamento em Classes de Equivalência [PRES02], Análise de Valores Limites e Grafo de Causa-Efeito [MYE79].
- ***Critérios Estruturais*.** Utiliza o código fonte e/ou a implementação para criação dos casos de teste, ou seja, caminhos lógicos do *software*, suas estruturas de dados e ciclos podem ser exercitados. Esta técnica pode garantir, por exemplo, que todas as decisões lógicas foram executadas pelo menos uma vez, a execução de todos os laços no seu limite e dentro do seu limite operacional, etc. São normalmente classificados em Critérios Baseados em Fluxo de Controle [RAP85], Critérios Baseados em Fluxo de Dados [RAP85, FRA88, MAL91] e Critérios Baseados na Complexidade [MCC76].
- ***Critérios Baseados em Defeitos*.** Deriva casos de teste com base nos defeitos inseridos com mais freqüência durante o processo de desenvolvimento de *software*. São critérios de teste baseados em defeitos a Semeadura de Defeitos [BUD81] e a Análise de Mutantes [DEM78]. O Critério de Análise de Mutantes é de particular interesse para este trabalho e será descrito mais detalhadamente.

4.2. Análise de Mutantes

A Análise de Mutantes é um critério de Teste Baseado em Defeitos [DEM78], onde um conjunto de teste T é adequado a P em relação a R, se para cada programa Q em R, ou Q é equivalente a P ou Q é diferente de P em, pelo menos, um caso de teste t em T.

Na prática, o critério é aplicado através da criação de um conjunto alternativo de programas chamados *mutantes* de P. Os mutantes diferem de P em apenas uma mudança sintática simples, determinada por um conjunto de operadores de mutação. Então, de acordo com a *Hipótese do Programador Competente* - supõe que os programas construídos estão bem próximos da sua forma correta - e o *Efeito do Acoplamento* - defeitos maiores são formados por uma composição de defeitos

menores e mais simples - um conjunto de teste T adequado ao conjunto de mutantes deverá ser capaz de revelar o defeito no programa P, descrito pelo mutante.

Para avaliar a adequação do conjunto de testes T, cada mutante, assim como o programa P, deve ser testado utilizando os casos de teste em T. Se o resultado do mutante Q é o mesmo do programa P, para todos os casos de teste em T, então Q é considerado um mutante vivo, caso contrário, é considerado um mutante morto.

Considere um conjunto de entradas E, S o conjunto de resultados obtidos devido ao processamento de P sobre o conjunto E. É dito que um mutante M é equivalente ao programa P se para todo conjunto E, M gera o mesmo conjunto S. Assim, um mutante vivo pode ser equivalente ao programa P, o que não nos dá nenhum dado importante, sendo assim descartado do conjunto de mutantes já que não contribui para adequação de T. A adequação dos testes é medida pelo escore de mutação MS(P,T), seguindo a regra abaixo.

$$MS(P,T) = \frac{N^{o}\ DE\ MUTANTES\ MORTOS}{N^{o}\ TOTAL\ DE\ MUTANTES\ GERADOS - N^{o}\ DE\ MUTANTES\ EQUIVALENTES}$$

Os operadores de mutação são funções que modificam o programa P em apenas uma mudança sintática simples, gerando mutantes deste programa a fim de revelar defeitos ainda não encontrados. Diversas implementações destes operadores estão disponíveis, como por exemplo, operadores de substituição de variáveis, substituição de operadores matemáticos, incremento, decremento, negação, entre muitas outras classificações. Na Figura 4.1 e na Figura 4.2 é mostrado um exemplo prático de como funcionam os operadores de mutação em geral. Considere um programa P mostrado na Figura 4.1 que informa o maior entre dois números passados como parâmetro. Têm-se então um exemplo de um programa genérico no formato original, ou seja, sem ter sofrido mutação. Depois da aplicação de um operador de mutação, neste caso um operador de substituição de operador relacional, a partir do programa P é gerado um mutante Q (Figura 4.2).

```
public int maior(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

Figura 4.1: Programa original não-mutante.

```
public int maior(int a, int b)
{
    if (a <= b)
        return a;
    else
        return b;
}
```

Figura 4.2: Programa mutante.

4.3. Teste de *Web Services*

Web Services têm o potencial de reduzir drasticamente a complexidade e os custos envolvidos em um projeto de integração de *software*. Contudo, o método de comunicação entre eles introduz grande complexidade nos quesitos de verificação e validação de *software* [OFF04].

A interação entre os *Web Services* se dá de duas formas distintas:

- ***Interação Ponto-a-Ponto.*** Quando é considerada a comunicação entre pares de componentes (Figura 4.3).
- ***Interação Multilateral.*** Quando a comunicação extrapola os limites da comunicação entre pares ocorrendo interação inter-pares. É definida como uma seqüência de mensagens entre múltiplos *Web Services* (Figura 4.4).

Figura 4.3: Interação ponto-a-ponto entre *Web Services* [OFF04].

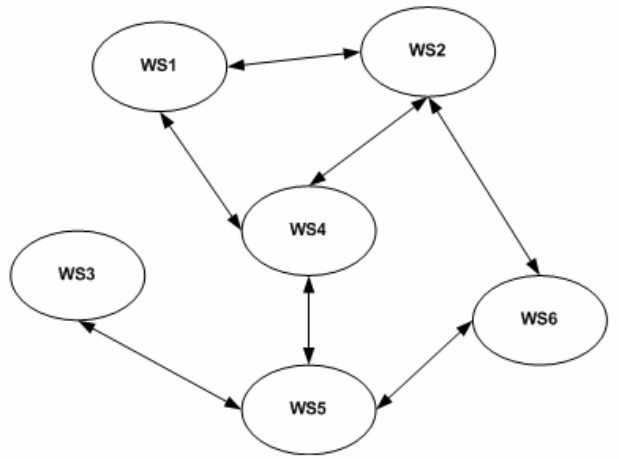

Figura 4.4: Interação multilateral entre *Web Services* [OFF04].

Acredita-se que nos próximos cinco anos a tecnologia de *Web Services* trará mudanças consideráveis na computação distribuída, mais especificamente, em relação às Arquiteturas Orientadas a Serviços (*Service Oriented Architecture* - SOA). Testar estas arquiteturas e os próprios *Web Services* constitui-se em um grande desafio para a disciplina de testes [BLO02a, DIL05].

Bloomberg [BLO02b] discute algumas questões relacionadas aos testes de *Web Services*, são elas:

- ***Teste das mensagens SOAP.*** Esta atividade envolve os testes dos mecanismos de requisição e resposta dos *Web Services* para mensagens SOAP.
- ***Teste de WSDL e utilização para planejamento de testes.*** A WSDL descreve uma interface para um serviço que estará disponível na Internet. Os testadores podem utilizar tais informações para ajudar no planejamento dos testes.
- ***Teste de UDDI.*** As três características fundamentais das Arquitetura Orientadas a Serviço são a capacidade de publicação, busca e amarração dos serviços. Um *Web Service* publica seu WSDL em um registro UDDI, onde os consumidores destes *Web Services* poderão encontrá-los. De uma perspectiva de testes, estas são novas funcionalidades a serem testadas.

- ***Teste de emulação de consumidores e produtores.*** Ferramentas de teste podem simular consumidores de *Web Services* enviando mensagens para estes, assim como podem simular produtores de *Web Services* retornando uma mensagem após um consumidor ter requisitado algum processamento.

Bloomberg [BLO02b] também discute sobre algumas outras questões envolvendo teste de *Web Services*, como por exemplo, relacionamentos síncronos e assíncronos, atores (*intermediaries*) SOAP, qualidade de serviço. Considerando estes aspectos alguns autores têm realizado pesquisas em testes de *Web Services*.

Em [TSA02], Tsai *et al.* descrevem um *framework* que converte especificações WSDL em cenários de teste. Lee *et al.* [LEE01] apresentam uma proposta de teste de sistemas que se comunicam através de mensagens XML, apesar de não serem mensagens SOAP. É apresentada uma técnica que utiliza Análise de Mutantes para validar a corretude semântica de interações entre componentes. Bultan *et al.* [BUL03] utilizam Máquinas de Estado Finito (*Finite State Machines)* para modelar e testar o comportamento de *Web* Services. Offutt and Xu [OFF04] propuseram um método de perturbação de dados para testar *Web Services*. Perturbação de dados consiste na modificação de mensagens com a utilização de operadores de mutação. Tais modificações assemelham-se ao que ocorre nos testes baseados na Análise de Mutantes [DEM78], contudo, a diferença está no fato de que os mutantes são, na realidade, os casos de teste a serem utilizados para testar o *Web Service*. Dois tipos de perturbações foram introduzidos: Perturbação de Dados e Perturbação de Interações.

Perturbação de Dados (*Data Value Perturbation* - DVP) modifica os valores dos dados da mensagem SOAP de acordo com regras definidas nos tipos dos valores e baseada na teoria de Teste de Valores Limites. Perturbação de Interação, por outro lado, modifica mensagens em comunicações RPC (RPC *Communication Perturbation* - RCP) e na comunicação de dados (*Data Communication Perturbation* - DCP). RCP é voltado para testar o uso dos dados e está dividido em dois tipos: uso de dados normais e uso de comandos SQL para indicar usos dos valores em programas e bancos de dados respectivamente. DCP é responsável por testar os relacionamentos e restrições dentro da mensagem.

É utilizado neste trabalho um modelo formal representado por uma gramática de árvore regular (*Regular Tree Grammar* - RTG). A RTG é uma 6-tupla <E, D, N, A, P, n_s > onde E é um conjunto finito de tipos de elementos; D é um conjunto finito de tipos de dados; N é um conjunto finito de não-terminais; A é um conjunto finito de tipos de atributos; P é um conjunto finito de regras de produção e n_s é o não-terminal inicial, $n_x \in N$.

Dado um conjunto de todos os elementos instâncias de N, um operador de mutação é dado por r = f (n_1, ..., n_i), onde f é uma função, i $\geq$ 1, cada n_1, ..., $n_i \in N$ e tem o mesmo tipo de dados, e r tem o mesmo tipo de dados que n_1, ..., n_i. Os seguintes operadores RPC foram definidos:

- ***Divide (n).*** Modifica o valor de *n* para 1 ÷ *n*, onde n é do tipo double.
- ***Multiply (n).*** Modifica o valor de *n* para *n* x *n*.
- ***Negative (n).*** Modifica o valor de *n* para -*n*.
- ***Absolute (n).*** Modifica o valor de n para |*n*|.
- ***Exchange (n_1, n_2).*** Substitui o valor de n_1 com n_2 e vice-versa, onde ambos n_1 e n_2 possuem o mesmo tipo.
- ***Unauthorized (str).*** Muda o valor da *string* str para str' OR '1' = '1, simulando técnicas de SQL *Injection* [SPE02].

Xu *et al.* [XUO05] propuseram um conjunto de operadores de perturbação para XML *Schemas* para criar mensagens inválidas. O XML *Schema* é representado por uma árvore e alguns operadores são definidos para inserir, remover e alterar nós ou sub-árvores no *schema* original. Mensagens XML são derivadas a partir dos documentos XML *Schema* alterados e enviados para os *Web Services* em teste, agindo como casos de teste. Os seguintes operadores foram definidos:

- ***InsertN.*** Insere um novo nó entre dois nós que estão conectados entre si.
- ***DeleteN.*** Remove um nó de uma árvore. Os nós-filhos são movidos para o nó-pai.
- ***InsertND.*** Insere um novo nó abaixo de outro nó.
- ***DeleteND.*** Remove um nó com seus tipos de dados.

- ***InsertT.*** Insere uma nova árvore abaixo de um determinado nó.
- ***DeleteT.*** Remove uma subárvore.
- ***ChangeE.*** Modifica as restrições de um XML *Schema*.

Tsai *et al.* [TSA04] propuseram uma técnica que utiliza a teoria de Testes Progressivos de Grupos para testar um grande número de *Web Services* disponíveis na Internet. Em um nível de teste unitário, os *Web Services* com as mesmas funcionalidades são testados em grupo usando um número progressivamente crescente de casos de teste. Duas fases de testes são definidas:

- ***Prescreening.*** Tem como objetivo eliminar os candidatos a *Web Services* menos prováveis a passarem para a próxima fase de testes. A sofisticação dos testes utilizados aumenta progressivamente, e os "vencedores" serão testados mais fortemente em fases subseqüentes. Para implementar este esquema, os *scripts* de teste são organizados em uma hierarquia de acordo com o relacionamento de dependência entre casos de teste.
- ***Runtime Group Testing.*** Os melhores candidatos identificados na fase anterior serão integrados ao sistema e testados em tempo de execução. Um sistema de votação é descrito, assumindo a posição de um oráculo para o sistema.

4.4. Considerações Finais

Neste capítulo foram apresentados diversos trabalhos relacionados a teste de aplicações baseadas na *web* e testes para *Web Services* em especial. Foi discutido sobre a importância da atividade de testes no que diz respeito ao controle de qualidade na produção de *software*, assim como sobre a importância e influência direta desta atividade para a qualidade do produto de *software* em desenvolvimento, e para a qualidade do processo utilizado.

A implementação de um critério de teste e sua utilização apoiada por uma ferramenta é de fundamental importância contribuindo para coleta de métricas e para que a atividade de testes e os conjuntos de casos de teste gerados sejam avaliados. Além disso, a automação de testes com o apoio de uma ferramenta

contribui fortemente para redução de custos desta atividade. Nos trabalhos [OFF04] e [XUO05] não há registro de que foi implementada alguma ferramenta para auxiliar o testador na aplicação da abordagem proposta, assim como os operadores propostos necessitam ser validados e refinados.

Esta pesquisa foi responsável pela implementação de uma ferramenta de testes capaz de apoiar o testador na aplicação do teste de perturbação, de forma a atender as deficiências apontadas anteriormente. Alguns operadores foram implementados na ferramenta de forma a permitir uma experimentação dos mesmos. O relato e análise dos resultados obtidos são mostrados no capítulo que segue.

"A noção de que boas técnicas restringem a criatividade
é como dizer que um artista pode pintar sem aprender
os detalhes de forma ou que um músico não
precisa conhecer teoria musical".
Marvin ZelKowitz.

5. Teste de Perturbação em *Web Services*

Este capítulo introduz um novo conjunto de operadores de perturbação de mensagens SOAP propostos neste trabalho, assim como a estrutura de *software* da ferramenta (SMAT-WS) produzida para apoiar a utilização dos operadores introduzidos e avaliar os operadores existentes.

5.1. Apresentação da SMAT-WS

Devido aos altos custos envolvidos com as atividades de teste de *software* [PRES02] é fundamental a utilização de ferramentas capazes de auxiliar os testadores no processo de planejamento, execução e controle de testes.

Para auxiliar a aplicação e avaliação dos operadores de perturbação para teste de *Web Services* foi desenvolvida uma ferramenta de teste, a SMAT-WS. Um exemplo de interação na qual a ferramenta estará envolvida está representado na Figura 5.1. Note que a ferramenta consegue atuar em cenários onde há a troca de mensagens SOAP entre sistemas, não importando a plataforma para quais foram desenvolvidos.

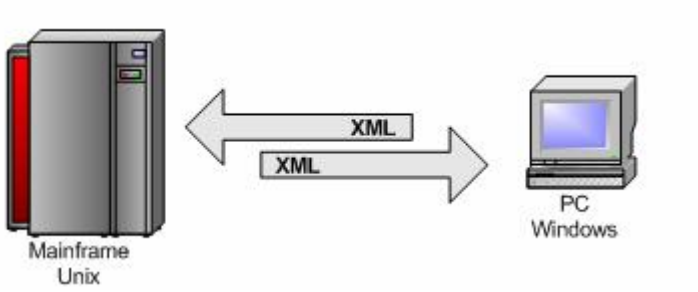

Figura 5.1: Interação entre sistemas em diferentes plataformas.

A SMAT-WS é uma ferramenta desenvolvida sobre a tecnologia Java, mais especificamente Java 2 *Standard Edition* (J2SE). Não necessita conexão a Internet

para ser utilizada, funcionando como uma aplicação *desktop*[1] convencional. A ferramenta utiliza alguns *frameworks* para desenvolvimento de aplicações, onde o principal deles é o Axis [AX00], *framework* construído em Java para desenvolvimento de aplicações baseadas na tecnologia de *Web Services*. Este *framework* é baseado nas licenças de *software* livre e está amplamente divulgado na comunidade de desenvolvimento de *Web Services*.

Com relação à tecnologia de *Web Services*, o sistema é capaz de executar os seguintes tipos de testes descritos na Seção 4.3:

- Testes de mensagens SOAP, compreendendo a modificação de uma mensagem SOAP dada e geração de casos de teste a partir da mesma com a utilização de operadores de perturbação.
- Utilização da linguagem de descrição de *Web Services* (WSDL) para planejamento dos testes. A ferramenta reduz a necessidade de o testador entender a linguagem, uma vez que fornece meios de geração automática de mensagens SOAP a partir da interpretação do descritor correspondente ao *Web Service* em teste.
- Emulação de consumidores de serviços.
- Testes de Regressão. Uma vez enviadas as requisições para os serviços correspondentes e recebidas as respostas, a SMAT-WS permite que os casos de teste sejam gravados a fim de servirem como teste de regressão em um momento futuro.

Além das características citadas, a ferramenta tem o objetivo de reduzir a necessidade de entendimento da tecnologia de *Web Services* por parte dos testadores. Assim os envolvidos nos testes desta tecnologia não precisam, necessariamente, ter conhecimentos avançados sobre *Web Services* para prosseguirem nas suas atividades. A SMAT-WS possui também uma arquitetura modularizada que facilita a extensão da ferramenta para atendimento de novos requisitos.

[1] Uma aplicação *desktop*, para o escopo deste trabalho, é definida como um software independente capaz de ser executado sem conexão a qualquer tipo de rede de comunicação.

5.2. Implementação

A ferramenta SMAT-WS implementa alguns dos operadores de perturbação de mensagens citados no capítulo anterior, assim como os operadores propostos neste trabalho e mostrados a seguir.

Como citado, a perturbação de dados é um método baseado na Análise de Mutantes, contudo, as mutações geradas sobre as mensagens são os próprios casos de teste a serem utilizados para verificar e validar, neste caso, *Web Services*.

Neste trabalho foi proposto um conjunto de operadores de perturbação de mensagens SOAP que estende o conjunto de operadores citado no capítulo anterior. Este novo conjunto de operadores foi definido para complementar e solucionar algumas deficiências verificadas quanto a utilização dos operadores até então existentes especificamente para tecnologia de *Web Services*.

Um operador de perturbação é definido como uma transformação SOAP, onde uma transformação são regras responsáveis por criar os casos de teste a partir de mensagens SOAP existentes, normalmente fornecidas pelo testador.

Considere uma mensagem SOAP uma árvore regular formada por um conjunto de nós N, onde n_1, ..., $n_i \in N$ e $i \geq 1$ e *nc* o número de nós-filhos do nó *n*. Ação(x) define a modificação e o resultado produzido pelo operador *x* e NMG(x) define o número total de mensagens modificadas geradas pelo operador *x*. Os operadores de perturbação propostos neste trabalho são definidos como segue.

5.2.1. Nulo (n)

Ação:

Anula o valor associado ao nó *n* na mensagem SOAP. Se não existir nenhum valor associado ao nó *n* este operador não produz nenhum efeito na mensagem.

NMG:

Este operador produz uma mensagem modificada para o valor associado ao nó *n* na mensagem SOAP.

Exemplo:

a)

```
<user>
  <name>Fred</name>
</user>

é modificada para:

<user>
  <name></name>
</user>
```

b)

```
<user>
  <name></name>
</user>

é modificada para:

<user>
  <name></name>
</user>
```

5.2.2. Incompleto (n)

Ação:

Apaga o nó *n* e seus nós-filhos da mensagem SOAP.

NMG:

Este operador gera uma mensagem modificada para o nó *n* na mensagem SOAP.

Exemplo:

a)

```
<user>
  <name>Fred</name>
</user>

é modificada para:

<user></user>
```

b)

```
<user>
  <name></name>
</user>

é modificada para:

<user></user>
```

5.2.3. Inversão (n)

Ação:

Inverte a ordem dos nós dentro do nó *n* em uma dada mensagem SOAP.

NMG:

Este operador gera *nc* - 1 mensagens para uma dada mensagem SOAP.

Exemplo:

```
<user>
  <name>Mary</name>
  <age>27</age>
</user>

é modificada para:

<user>
  <age>27</age>
  <name>Mary</name>
</user>
```

5.2.4. Inversão de Valores (n)

Ação:

Inverte os valores associados aos nós-filhos do nó *n* em uma dada mensagem SOAP.

NMG:

Este operador gera *nc* - 1 mensagens para uma dada mensagem SOAP.

Exemplo:

```
<user>
  <name>Mary</name>
  <age>27</age>
</user>

é modificada para:

<user>
  <name>27</name>
  <age>Mary</age>
</user>
```

5.2.5. Tamanho (n)

Ação:

Altera o tamanho do valor associado ao nó *n* em uma dada mensagem.

NMG:

Este operador gera dois valores modificados: (1) transforma o valor associado ao nó *n* para valor + '$'; (2) remove o último caractere do valor associado ao nó *n*.

Exemplo:

a)

```
<user>
  <name>Mary</name>
</user>

é modificada para:

<user>
  <name>Mar</name>
</user>
```

b)

```
<user>
  <name>Mary</name>
</user>

é modificada para:

<user>
  <name>Mary$</name>
</user>
```

5.2.6. Espaço (n)

Ação:

Modifica para ‘ ‘ (espaço em branco) o valor associado ao nó *n* em uma dada mensagem.

NMG:

Este operador gera uma mensagem modificada para o valor associado ao nó *n* em uma dada mensagem SOAP.

Exemplo:

```
<user>
  <name>Fred</name>
</user>

é modificada para:

<user>
  <name> </name>
</user>
```

5.3. Arquitetura da Ferramenta

A SMAT-WS é formada por um conjunto de módulos, como mostrados na Figura 5.2, responsáveis por funções próprias dentro da cadeia de execução do sistema. O sistema foi construído sobre os conceitos da teoria de Orientação a Objetos, onde cada módulo comunica-se com o seu sucessor através do envio de mensagens, desta forma é possível definir claramente as interfaces entre eles. Suas funcionalidades serão detalhadas na Seção 5.3.1.

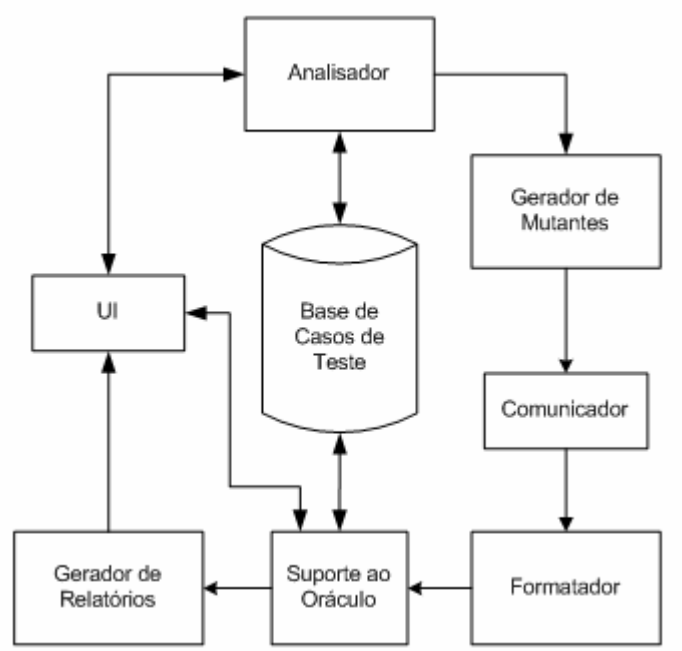

Figura 5.2: Arquitetura SMAT-WS.

A arquitetura do sistema está estruturada de forma a atender dois momentos importantes do fluxo de informações entre *Web Services*: Requisição e Resposta. De forma bem abrangente, a Figura 5.3 demonstra o contexto no qual a SMAT-WS estará inserida. Simulando um *Web Service*, a ferramenta envia uma mensagem de requisição de serviço para um outro *Web Service* B, e em seguida, recebe uma mensagem contendo o resultado da execução do serviço pedido do mesmo *Web Service*. Será detalhado adiante como cada módulo está envolvido em cada momento do fluxo de informações.

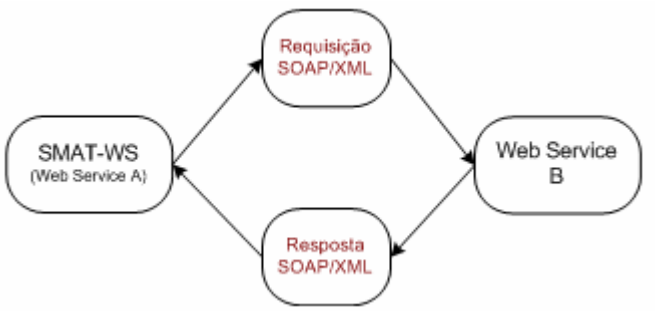

Figura 5.3: Simulação de um *Web Service* através da SMAT-WS.

5.3.1. Módulos Integrantes

Nesta seção serão descritos em detalhes os módulos que compõem a arquitetura SMAT-WS.

5.3.1.1. UI

Responsável por gerar e controlar a interface entre a ferramenta de testes e o usuário. Possui conexão com três pontos importantes da arquitetura do sistema, a saber:

- ***Suporte ao Oráculo.*** Fornece meios para o testador decidir sobre a corretude das respostas obtidas em relação às respostas esperadas.
- ***Analisador.*** Fornece a interface necessária para obtenção da mensagem SOAP original que será fonte de todos os casos de teste gerados, seja essa entrada feita de forma manual ou automática via WSDL.
- ***Gerador de Relatório.*** Fornece informações para o usuário verificar os resultados obtidos durante o procedimento de teste.

5.3.1.2. Analisador

O Analisador é o módulo responsável por verificar a corretude das mensagens SOAP informadas e que serão utilizadas como fonte de perturbação de dados para geração dos casos de teste, mensagens SOAP modificadas.

Também possui o papel de interpretador de descritores de *Web Services* - WSDL. Com isso, o Analisador é responsável por auxiliar o testador na criação de mensagens SOAP para o *Web Service* em teste. Esta funcionalidade tem como proposta reduzir o nível de conhecimento técnico necessário para que um testador possa executar procedimentos de teste sobre a tecnologia de *Web* Services. Contudo, para esta versão da ferramenta, esta facilidade está adequada apenas para *Web Services* que seguem o estilo de passagem de mensagens no formato RPC.

5.3.1.3. Gerador de Mutantes

O Gerador de Mutantes é o principal módulo de todo o sistema. É responsável por gerar as mensagens SOAP modificadas a partir de uma dada mensagem SOAP original. Essas mensagens modificadas serão utilizadas posteriormente como casos de teste para os *Web Services*.

Este módulo implementa alguns dos operadores de perturbação mostrados na Tabela 5.1, responsáveis pelas mutações nas mensagens. Este módulo está definido de forma extensível, onde novos operadores podem ser implementados facilmente e integrados ao sistema.

Tabela 5.1: Listagem dos operadores de perturbação.

Operador	Implementado	Introduzido em
memberOf		[LEE01]
lenOf	***	[LEE01]
Divide	✓	[OFF04]
Multiply	✓	[OFF04]
Negative	✓	[OFF04]
Absolute	✓	[OFF04]
Exchange	***	[OFF04]
Unauthorized	✓	[OFF04]
InsertN		[XUO05]
DeleteN		[XUO05]
InsertND		[XUO05]
DeleteND		[XUO05]
InsertT		[XUO05]
DeleteT	***	[XUO05]
ChangeE		[XUO05]
Boundary	***	[BEI90]
Nulo	✓	Seção 5.2
Incompleto	✓	Seção 5.2
Inversão	✓	Seção 5.2
Inversão de Valores	✓	Seção 5.2
Tamanho	✓	Seção 5.2
Espaço	✓	Seção 5.2

*** Este operador foi implementado de forma modificada.

Como pode ser observado a ferramenta implementa todos os operadores propostos na Seção 5.2 e para permitir comparação e avaliação, também foram implementados outros operadores apresentados na literatura e descritos no Capítulo 4. Observa-se também que alguns desses operadores foram implementados de forma diferente para atender deficiências percebidas durante algumas experiências

realizadas durante esta pesquisa. Na Tabela 5.2 são mostradas as siglas que serão utilizadas para referenciar os operadores implementados no restante deste trabalho. Dentre os operadores modificados (vide Tabela 5.1) encontram-se:

- ***LenOf.*** As modificações nas regras deste operador deram origem ao operador Tamanho.
- ***Exchange.*** As modificações nas regras deste operador deram origem ao operador Inversão de Valores. Sua versão original troca os valores entre elementos com o mesmo tipo de dados.
- ***DeleteT.*** As modificações nas regras deste operador deram origem ao operador Incompleto. Sua versão original remove uma subárvore da representação em árvore de um documento XML contanto que a subárvore contenha pelo menos um nó-filho.
- ***Boundary.*** As modificações nas regras deste operador deram origem ao operador Boundary, baseado em testes de valores limites [BEI90] e Boundary Extended que cria os casos de teste mesmo quando não se sabe o tipo do parâmetro da mensagem SOAP utilizada como teste.

Tabela 5.2: Legenda de operadores implementados.

Tamanho	**TA**
Incompleto	**I**
Nulo	**N**
Boundary Extended / Valor Limite Estendido	**VLE**
Inversão	**IN**
Inversão de Valores	**INV**
Espaço	**E**
Boundary / Valor Limite	**VL**
Unauthorized	**U**
Divide	**D**
Multiply	**M**
Negative	**NE**
Absolute	**AB**

5.3.1.4. Comunicador

Este módulo é responsável por toda a comunicação entre a SMAT-WS e os *Web Services* em teste pelo sistema. A comunicação implica envio de mensagens de requisição e recebimento das mensagens de resposta.

Durante o recebimento das mensagens de resposta, este módulo também é responsável por repassá-las para os demais módulos do sistema para continuação do fluxo de tratamento da mensagem.

5.3.1.5. Suporte ao Oráculo

Este módulo é responsável por auxiliar o testador na análise e comparação dos resultados esperados com os resultados obtidos. O Suporte ao Oráculo possui duas formas básicas de atuação, uma delas via a interface com o usuário, permitindo o controle e verificação da corretude das mensagens que estão sendo enviadas como resposta no sistema. Outra forma, mais automática, permite que a Base de Casos de Teste forneça uma mensagem resposta esperada para o módulo permitindo a automatização do processo.

O Suporte ao Oráculo também é responsável pela coleta de métricas durante o processo de validação dos casos de teste pelo testador. Estas informações são em seguida repassadas para o módulo Gerador de Relatórios, responsável pela apresentação das mesmas.

5.3.1.6. Formatador

Com o alto número de mensagens modificadas geradas pelos operadores, é necessária uma forma de facilitar a atividade dos testadores. Cada mensagem recebida como resposta dos *Web Services* em teste é verificada quanto a sua corretude. De forma genérica, o Formatador é capaz de transformar a representação da mensagem recebida e repassada pelo Comunicador em um formato padrão e de fácil entendimento pelo usuário, facilitando o processo de teste.

Após a execução dos testes, o testador tem a opção de manter na Base de Casos de Teste todas as informações relativas ao procedimento recém executado. De posse destas informações, o procedimento de teste pode ser reexecutado a qualquer momento. O Formatador, neste contexto, atua como analisador de mensagens SOAP, comparando mensagens obtidas com as mensagens esperadas com a utilização de uma representação em árvore das mensagens. O módulo

verifica a diferença entre os nós desta árvore formatando o resultado da comparação para utilização pelo testador.

5.3.1.7. Gerador de Relatórios

Responsável pela geração e formatação de relatórios contendo informações dos resultados conseguidos em relação à bateria de testes efetuada, como: número de casos de teste que revelaram defeitos, número de casos de teste que não revelaram defeitos, número de testes efetuados, número de casos de teste por operador, tempo de criação dos testes, entre outros tipos de resultados. Cabe salientar que todas estas informações disponíveis foram coletadas e enviadas pelo Suporte ao Oráculo durante o processo de teste.

5.3.1.8. Base de Casos de Teste

Este módulo é responsável por controlar o repositório de casos de teste do sistema. Este repositório armazena informações, como:

- ***Informações de Requisição.*** São as mensagens modificadas (mutantes) que serão utilizadas pelo Analisador para submeter novamente os casos de teste que irão servir como testes de regressão.
- ***Informações de Resposta.*** São as mensagens de resposta esperadas utilizadas pelo Suporte ao Oráculo a fim de comparar os resultados obtidos com os resultados fornecidos por este módulo.

5.4. Funcionamento

Para melhorar o entendimento do sistema, serão explicados com maiores detalhes os fluxos e as conexões entre os módulos da ferramenta. Como citado anteriormente, a Figura 5.3 mostra a atuação da SMAT-WS em conjunto com um dado *Web Service* B a ser testado.

A Figura 5.4 mostra os módulos envolvidos durante um pedido de requisição de serviço para um *Web Service* qualquer. O testador, através do módulo de interface com o usuário (UI), informa uma mensagem SOAP inicial que servirá como fonte

para perturbações e geração das mensagens SOAP modificadas. A mensagem SOAP fornecida pelo testador pode também ser obtida a partir do WSDL que descreve o *Web Service*. A mensagem SOAP da Figura 5.5 será utilizada até o final da seção, o que facilitará a compreensão dos exemplos.

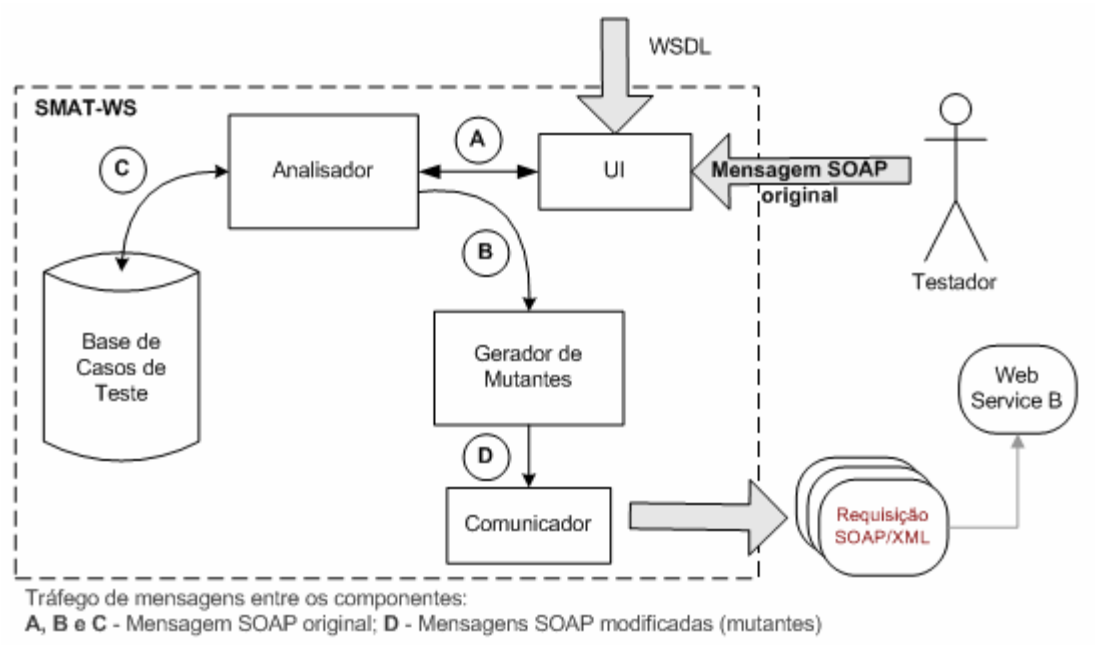

Figura 5.4: Módulos envolvidos durante a requisição.

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:buscarProduto
      xmlns:ns1="http://soapinterop.org/">
        <codigo_produto xsi:type="xsd:string">AB1234</codigo_produto>
    </ns1:buscarProduto>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 5.5: Pedido de requisição SOAP.

Após informar a mensagem SOAP inicial, o módulo Analisador valida a mensagem e repassa para o módulo Gerador de Mutantes responsável por efetuar as modificações na mensagem encaminhada para ele. Após a execução das perturbações algumas dezenas de mensagens SOAP serão criadas. Um exemplo de mensagem SOAP modificada está mostrado na Figura 5.6.

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:buscarProduto
      xmlns:ns1="http://soapinterop.org/">
        <codigo_produto xsi:type="xsd:string">AAAAAAAAAAAA</codigo_produto>
    </ns1:buscarProduto>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 5.6: Exemplo de mensagem SOAP modificada a partir da utilização de um operador de perturbação.

Cada uma das mensagens geradas pelo Gerador de Mutantes é repassada então para o módulo Comunicador que tem como responsabilidade transmitir todos os casos de teste para o *Web Service* B em teste.

Depois de efetuada a transmissão, a SMAT-WS receberá as dezenas de mensagens contendo a resposta da execução do serviço requisitado para cada uma das mensagens enviadas como mostrado na Figura 5.7. Um exemplo de resposta do *Web Service* B em teste está mostrado na Figura 5.8.

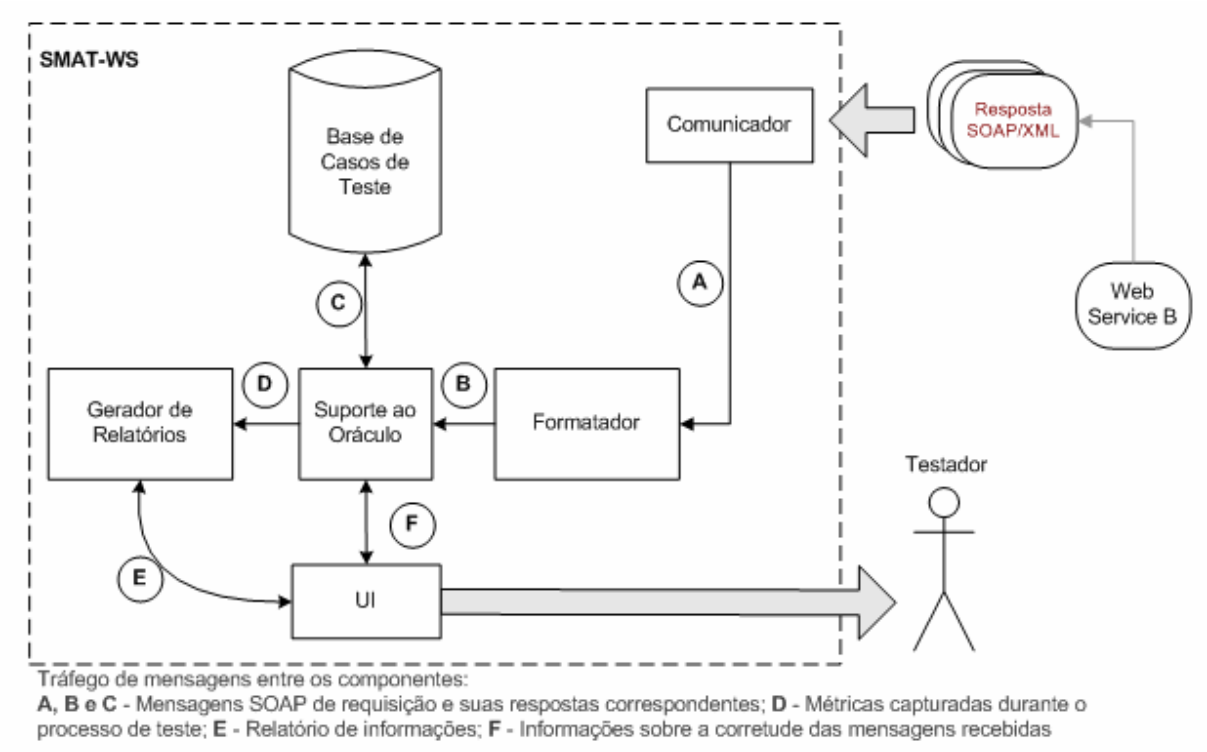

Figura 5.7: Módulos envolvidos durante a resposta.

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:buscarProdutoResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://soapinterop.org/">
      <codigo_produto xsi:type="xsd:string">AB1234</codigo_produto>
      <nome_produto xsi:type="xsd:string">Liquidificador</nome_produto>
      <qte_estoque xsi:type="xsd:int">532</qte_estoque>
      <valor_unitario xsi:type="xsd:float">49.99</valor_unitario>
    </ns1:buscarProdutoResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 5.8: Mensagem SOAP de resposta enviada por um *Web Service*.

O módulo Comunicador é responsável pela tarefa de recepção das mensagens de retorno do *Web Service* em teste. Depois de recepcionadas, as mensagens trafegam diretamente para o módulo Formatador.

Percebe-se que, à medida que as mensagens vão aumentando de tamanho, o que é o caso de *Web Services* mais complexos que o mostrado no exemplo em questão, torna-se inviável a tarefa do testador em decidir sobre a corretude das mensagens de resposta recebidas. Este problema é amenizado pelo módulo Formatador, pois, uma de suas responsabilidades é prover ao módulo Suporte ao Oráculo uma formatação mais simples e de fácil verificação para as mensagens encaminhadas. Um exemplo de mensagem SOAP formatada para o exemplo da Figura 5.8 está mostrado na Figura 5.9.

```
codigo_produto: AB1234
nome_produto: Liquidificador
qte_estoque: 532
valor_unitario: 49.99
```

Figura 5.9: Exemplo de mensagem SOAP da Figura 5.8 após formatação do módulo Formatador.

Em seguida, depois de recepcionadas pelo Comunicador e transformadas pelo Formatador, o módulo de Suporte ao Oráculo é responsável por auxiliar o testador nas decisões sobre corretude das mensagens, como citado anteriormente. O testador, com base na especificação do *software* e de posse dos resultados esperados para cada teste realizado, informa para a ferramenta sobre cada resultado obtido durante o procedimento.

À medida que o testador interage com o Suporte ao Oráculo, a ferramenta captura métricas (número de casos de teste que revelaram defeitos, tempo total de execução, número total de casos de teste gerados por operador, etc.) sobre o resultado dos testes executados. Estas serão enviadas ao Gerador de Relatórios, responsável por demonstrá-las de forma automatizada para o usuário.

Com isso, encerra-se a execução do procedimento de teste. A partir de então o testador poderá reexecutar qualquer um dos casos de teste disponíveis; salvar na Base de Casos de Teste todos os casos de teste gerados bem como os resultados obtidos, servindo como testes de regressão posteriormente, ou ainda, obter da Base de Casos de Teste qualquer um dos testes executados para ser reexecutado. O Apêndice A contém exemplos de interação entre o usuário e a ferramenta.

5.5. Considerações Finais

Neste capítulo foi apresentado um novo conjunto de operadores de perturbação para mensagens SOAP e foi descrita a SMAT-WS, ferramenta de apoio a testes de *Web Services* que implementa diversos operadores de perturbação. A ferramenta é capaz de executar testes de mensagens SOAP, utilizar o descritor de *Web Services* (WSDL) para planejamento e projeto de testes, emular consumidores de serviços oferecidos na *web* e executar testes de regressão.

A tecnologia de *Web Services* é formada por um conjunto de protocolos e especificações que tendem a dificultar tanto a produção quanto a validação e verificação dos mesmos. A ferramenta proposta, além de auxiliar diretamente nas atividades de verificação e validação de *Web Services,* reduz a necessidade de competência técnica exigida para testar *softwares* desenvolvidos com esta tecnologia. Contudo, para esta versão da ferramenta, apenas interações entre pares

de serviços pode ser testada, não envolvendo formas de atender demandas como comunicação multilateral entre eles na presente implementação.

Os operadores introduzidos possuem algumas características particulares, como:

- Não existe dependência com XML *Schemas* para geração das mensagens modificadas que serão utilizadas como casos de teste;
- As mensagens SOAP são diretamente modificadas pelos operadores de perturbação e por regras específicas do protocolo SOAP, o que causa uma redução no tempo de execução dos procedimentos de teste. A geração direta de casos de teste a partir de mensagens SOAP tende a criar um número menor de casos de teste comparada a abordagens de perturbação de mensagens a partir de gramáticas como XML *Schema*, o que impacta diretamente na redução dos custos com a atividade de teste.

No capítulo seguinte serão mostrados os resultados e análises do estudo de caso realizado com o apoio da ferramenta e dos operadores de perturbação implementados.

"Erros são mais comuns, mais
penetrantes e mais problemáticos no
software do que em outras tecnologias".
David Parnas

6. Estudo de Caso

Este estudo de caso teve como objetivo avaliar a ferramenta construída quanto ao apoio ao testador, assim como atestar sua contribuição para a disciplina de teste de *Web Services* e explorar a utilização dos operadores de perturbação de mensagens propostos.

Foram selecionados nove *Web Services* que fazem parte de um sistema responsável por integrar diversas aplicações desenvolvidas em plataformas diferentes. Todos os *Web Services* foram submetidos aos casos de teste, mensagens SOAP modificadas a partir da utilização dos operadores de perturbação produzidos pela ferramenta SMAT-WS. Em seguida foram coletados os resultados obtidos e tabulados a fim de se estabelecer uma forma simples de analisar as informações. Os seguintes parâmetros foram considerados para cada *Web Service* em avaliação:

- Tempo total de geração dos casos de teste;
- Número total de casos de teste gerados para cada *Web Service*;
- Número de casos de teste gerados por operador;
- Número total de defeitos encontrados para cada *Web Service*;
- Número de defeitos encontrados para cada *Web Service* por operador.

6.1. Procedimento de Teste

Nesta seção será descrita a abordagem utilizada para obtenção dos resultados encontrados com o estudo de caso. As informações obtidas no processo foram analisadas e derivaram diversas conclusões sobre a efetividade da ferramenta produzida em conjunto com os operadores de perturbação de mensagens utilizados.

Os verdadeiros nomes das aplicações e alguns detalhes de implementação foram omitidos por questões que envolvem acordos de não-divulgação das informações contidas no sistema.

6.1.1. *Web Services* Avaliados

O sistema utilizado como estudo de caso é um sistema integrador de aplicações (*WS.2* mostrado na Figura 6.1) que auxiliam a administração do governo federal. Ele serve como intermediário entre os mais diversos *softwares* da administração pública, permitindo a troca de informações entre tais sistemas, o que até então não acontecia.

A arquitetura do sistema encontra-se especificada de forma a interagir com diversos outros sistemas estruturadores como mostrado na Figura 6.1. O *WS.2* é formado por um conjunto de nove *Web Services* que podem ser acessados por qualquer um dos sistemas estruturadores (Aplicações W, X e Y na Figura 6.1) de âmbito nacional, fornecendo um serviço integrado de troca de informações. Os sistemas estruturadores são aplicações que centralizam informações sobre uma determinada estrutura governamental, como órgãos públicos no caso da Aplicação Z ou informações sobre recursos humanos no caso da Aplicação W, por exemplo. Cada uma destas aplicações e *Web Services* fazem parte de uma lógica de negócio complexa e de criticidade considerável no processo de integração, sendo de extrema importância o funcionamento adequado e dentro dos padrões de qualidade pré-estabelecidos durante o seu desenvolvimento. Desta forma, fez-se necessário que os quesitos de qualidade requeridos fossem atestados por uma atividade de teste rígida, definida e planejada, garantindo a obtenção de resultados satisfatórios neste processo. Este contexto foi fundamental para a utilização deste sistema como uma forma de experimentar as idéias propostas neste trabalho.

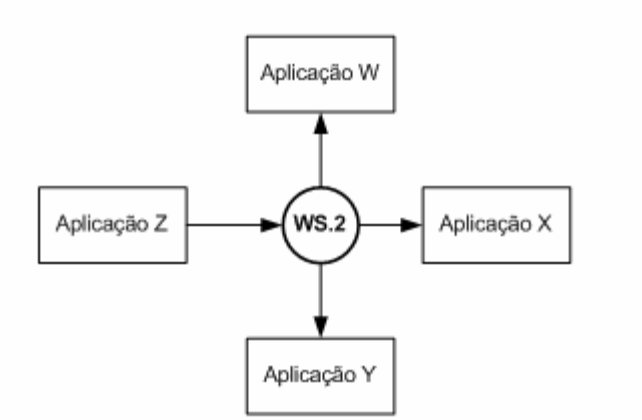

Figura 6.1: Integração do WS.2 com demais sistemas.

O conjunto de *Web Services WS.2* foi planejado para solucionar problemas de integração entre sistemas, como comentado anteriormente, contudo, nesta etapa do projeto, apenas informações de órgãos governamentais estão sendo integradas, ou seja, o *WS.2* funciona como integrador da Aplicação Z para os demais sistemas integradores, sendo estes apenas agentes requisitantes de informações (eventos). O restante das integrações será implementado em momentos futuros e mais oportunos na aplicação pela empresa.

Dentro deste contexto serão descritos cada *Web Service* que compõe o *WS.2*.

- ***obterDadosOrgao***. Responsável por obter os dados relativos a um determinado órgão governamental que se encontram cadastrados em outro sistema controlador da estrutura de órgãos – Aplicação Z (Figura 6.1).

- ***abrirListaEventosPendentes***. Abre uma lista de eventos que ainda encontram-se pendentes para um determinado órgão. Um evento é definido no escopo deste trabalho como uma modificação estrutural na hierarquia de entidades. Estas entidades podem ser órgãos, pessoas, etc. Este *Web Service* está representado como A na Figura 6.2.

- ***obterEventoDaListaPendencia***. Obtém os dados de um evento em específico dentro da lista de eventos criada com o apoio do *Web*

Service anterior. Este *Web Service* está representado como B na Figura 6.2.

- ***obterDadosEvento****.* Obtém detalhes sobre um determinado evento desejado. Este *Web Service* está representado como C na Figura 6.2.

- ***efetivarIntegracaoEventoSiorg****.* A efetivação da integração consiste na obtenção e posterior atualização da base de dados do *WS.2* pelo sistema estruturador responsável pelo evento a ser consumido. Este *Web Service* está representado como D na Figura 6.2.

- ***fecharListaEventosPendentes****.* Informa que o processo de integração será fechado ou concluído. Este *Web Service* está representado como E na Figura 6.2.

- ***obterDeParaOrgao****.* O *WS.2* informa através deste *Web Service* a correlação entre os diversos sistemas estruturadores. Em resumo, ele é uma tabela De-Para responsável por conter os mapeamentos entre os eventos para cada sistema.

- ***atualizaDeParaOrgao****.* Responsável por atualizar a base de dados contendo os mapeamentos entre os eventos para cada sistema.

- ***incluirEventoProduzido****.* A Aplicação Z alimenta o *WS.2* com os eventos produzidos por aquela aplicação. É através deste *Web Service* que se inicia o processo de integração entre os sistemas.

Estes são os *Web Services* integrantes do *WS.2* e suas respectivas funcionalidades, embora não esteja explícito como estes *Web Services* interagem entre si para executar as tarefas de integração entre os sistemas.

Como citado anteriormente, o *WS.2* é formado por nove *Web Services*. Quatro deles são independentes entre si e podem ser acessados por outros aplicativos de forma assíncrona. Cinco deles são dependentes entre si e seguem o fluxo de informações mostrado na Figura 6.2. Percebe-se que existe uma sincronia entre os

Web Services e que estes não podem executar suas funcionalidades de forma paralela, uma vez que os parâmetros de saída de um *Web Service* são os parâmetros de entrada para o posterior, formando assim um encadeamento lógico das informações que trafegam entre eles.

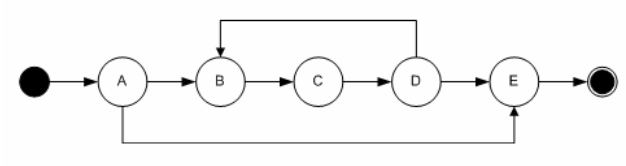

Figura 6.2: *Web Services* integrantes do *WS.2*.

6.1.2. Metodologia

A metodologia adotada nesta pesquisa foi definida de forma que os nove *Web Services* fossem testados a partir da utilização e apoio da ferramenta proposta neste trabalho. Foram definidos alguns parâmetros para permitir a análise dos resultados obtidos, como: eficiência dos operadores, percentagem de defeitos encontrados e o número de casos de teste ineficientes.

Inicialmente foi gerada uma mensagem SOAP válida para cada *Web Service* em teste que deu origem a todos os mutantes para todos os operadores. Estas nove mensagens foram submetidas à ferramenta para geração das mensagens modificadas. Após a geração automática das mutações as mensagens SOAP modificadas foram submetidas aos *Web Services* em teste a fim de revelarem defeitos. Cada defeito encontrado foi registrado em conjunto com o respectivo operador de perturbação que o revelou para que fossem extraídas as métricas desejadas. Alguns dos defeitos foram revelados mais de uma vez, ou seja, diferentes operadores de perturbação descobriram o mesmo defeito para um dado componente de *software* em teste. Contudo, foram registradas todas as vezes que cada defeito aparecia, construindo um conjunto de informações independentes para cada operador.

Igualmente, ocorreram casos onde um operador de perturbação revelou o mesmo defeito, para um dado componente de *software* em teste, mais de uma vez, ou seja, foram gerados casos de teste iguais. Contudo, neste caso, a contagem foi apenas para a primeira vez que o defeito apareceu.

O testador teve atuação em dois momentos fundamentais durante todo o processo: geração da mensagem SOAP que dá origem as mensagens mutantes e na atuação como oráculo, informando à ferramenta quais casos de teste geraram resultados obtidos iguais aos resultados esperados.

Os nove *Web Services* foram submetidos individualmente aos casos de teste gerados, assim como existiu total acesso as bases de dados utilizadas, facilitando ao testador que conduziu o experimento verificar a corretude dos *Web Services* até o nível de gravação dos dados, caso necessário.

O processo de construção dos *Web Services* foi acompanhado desde os momentos iniciais da especificação de projeto. Assim, pôde-se avaliar de forma natural, ou seja, sem introdução de defeitos propositais, os defeitos encontrados com maior freqüência na produção de *Web Services*.

O ambiente de produção para os *Web Services* era composto por um conjunto de plataformas, sistemas operacionais e linguagens de programação diferentes, exercitando uma das principais características da tecnologia: integração de projetos de *software* envolvendo ambientes distribuídos e heterogêneos.

6.1.3. Resultados Obtidos

Os defeitos encontrados foram registrados e relatados em conjunto com os operadores que geraram os casos de teste que os revelaram. Os dados coletados a partir da submissão dos casos de teste gerados deram origem aos resultados que seguem. A Tabela 6.1 informa o número de casos de teste gerados por operador e para cada *Web Service* testado. São utilizadas as siglas apresentadas na Tabela 5.2. Os operadores numéricos propostos por Offutt *et al.* [OFF04] não estão presentes neste trabalho pelo fato da natureza dos *Web Services* testados. Por razões históricas, os *Web Services* usados no experimento não utilizam tipos numéricos para troca de informação. Todos os dados são passados como sendo do tipo *string*. Na realidade, este fato apontou alguns problemas a serem validados no que diz respeito à *Web Services*, como por exemplo, a validação dos dados do lado servidor, uma vez que toda a informação é tratada como *string*.

Tabela 6.1: Número de casos de teste gerados por operador.

***Web* Service**	**TA**	**I**	**N**	**VLE**	**IN**	**INV**	**E**	**VL**	**U**
abrirListaEventosPendentes	12	6	6	42	5	5	6	18	6
obterEventoDaListaPendencia	10	5	5	34	4	4	5	15	5
obterDadosEvento	8	4	4	26	3	3	4	12	4
efetivarIntegracaoEventoSiorg	14	7	7	41	4	6	7	21	7
fecharListaEventosPendentes	8	4	4	26	3	3	4	12	4
incluirEventoProduzido	18	9	9	54	8	8	9	27	9
obterDadosOrgao	8	4	4	26	3	3	4	12	4
obterDeParaOrgao	12	6	6	48	5	5	6	18	6
atualizaDeParaOrgao	18	9	9	66	8	8	9	27	9
Total	**108**	**54**	**54**	**363**	**43**	**45**	**54**	**162**	**54**

A Tabela 6.2 informa o número de defeitos encontrados por operador e para cada *Web Service* testado. É importante clarificar que o somatório destes defeitos não retrata o resultado exato do experimento, uma vez que o mesmo defeito foi encontrado por diversos operadores, portanto, registrado diversas vezes nesta tabela.

Tabela 6.2: Número de defeitos encontrados por operador.

***Web* Service**	**TA**	**I**	**N**	**VLE**	**IN**	**INV**	**E**	**VL**	**U**
abrirListaEventosPendentes	1	1	1				2		
obterEventoDaListaPendencia			1		1	1		5	
obterDadosEvento	2	2	2	4	1	1	2	2	2
efetivarIntegracaoEventoSiorg	3	3	4	7		1	4	4	4
fecharListaEventosPendentes									
incluirEventoProduzido								3	
obterDadosOrgao									
obterDeParaOrgao	6	5	6	6			6	6	5
atualizaDeParaOrgao	4	5	7	7		4	5	7	5
Total	**16**	**16**	**21**	**24**	**2**	**7**	**19**	**27**	**16**

Na Tabela 6.3 é mostrado o número de casos de teste que revelaram defeitos, independente deste defeito já ter sido revelado anteriormente. Estes casos de teste são definidos, para o escopo deste trabalho, como casos de teste eficientes.

Tabela 6.3: Casos de teste eficientes por operador.

Web Service	TA	I	N	VLE	IN	INV	E	VL	U
abrirListaEventosPendentes	2	1	1				2		
obterEventoDaListaPendencia			1		4	1		9	
obterDadosEvento	4	2	2	12	3	1	2	2	2
efetivarIntegracaoEventoSiorg	5	3	4	25		2	4	4	4
fecharListaEventosPendentes									
incluirEventoProduzido								3	
obterDadosOrgao									
obterDeParaOrgao	11	5	6	36			6	6	5
atualizaDeParaOrgao	6	5	7	32		4	5	7	5
Total	**28**	**16**	**21**	**105**	**7**	**8**	**19**	**31**	**16**

Na Tabela 6.4 foram condensados alguns indicadores importantes para a análise do experimento de uma forma mais abrangente. Os mais importantes são:

- ***Eficiência.*** A eficiência do operador é definida como a razão entre o número de defeitos encontrados para o operador dividido pelo número de casos de teste gerados para o mesmo operador. Tem a função de indicar quais dos operadores obtiveram melhores resultados no experimento. A eficiência é um parâmetro adequado, uma vez que considerar apenas o número de casos de teste gerados ou o número de defeitos encontrados separadamente não fornece informação suficiente para análise sobre os operadores.
- ***% Defeitos Encontrados.*** Este indicador define a porcentagem de defeitos encontrados por cada operador. É encontrado através da razão entre o número de defeitos encontrados pelo operador dividido pelo número total de defeitos encontrados.
- ***% Casos de Teste Ineficientes.*** Este indicador informa a porcentagem de casos de teste que revelaram o mesmo defeito para o mesmo operador. Sua interpretação relata quantos dos casos de teste gerados revelaram o mesmo defeito. É encontrado através da expressão: $1-\frac{TE}{CTE}$, onde TE é o número de defeitos encontrado por operador e CTE o número de casos de teste eficientes por operador (Tabela 6.3).

Tabela 6.4: Resultados Obtidos.

	TA	I	N	VLE	IN	INV	E	VL	U
Total de Casos de Teste	108	54	54	363	43	45	54	162	54
Total de Defeitos Encontrados	16	16	21	24	2	7	19	27	16
Total de Casos de Teste Eficientes	28	16	21	105	7	8	19	31	16
Eficiência	14,81%	29,63%	38,89%	6,61%	4,65%	15,56%	35,19%	16,67%	29,63%
Defeitos Encontrados	32,65%	32,65%	42,86%	48,98%	4,08%	14,29%	38,78%	55,10%	32,65%
Casos de Teste Ineficientes	42,86%	0,00%	0,00%	77,14%	71,43%	12,50%	0,00%	12,90%	0,00%

Durante todo o experimento foram registrados 49 defeitos de tipos diferentes com a utilização dos diversos operadores de perturbação. Os defeitos variaram de defeitos de lógica de negócio a validação de campos de dados. Cabe salientar que o experimento foi realizado durante o processo de desenvolvimento dos *Web Services*, não havendo defeitos semeados. Todos os defeitos encontrados foram defeitos gerados ao longo do desenvolvimento.

De acordo com o indicador de eficiência de operadores, existe um indicativo de que os operadores *Nulo (N), Espaço (E)* e *Unauthorized (U)* são, de fato, eficientes devido aos resultados conseguidos e relatados na Tabela 6.4. Por outro lado, o operador *Valor Limite Estendido (VLE)* apesar de revelar uma boa porcentagem dos defeitos (48,98%) não é um operador tão eficiente devido ao número de casos de teste gerados. Ainda, a porcentagem de casos de teste ineficientes gerados é bastante alta. A conclusão que se chega é que este operador pode se tornar um operador mais eficiente com alguns ajustes em suas regras de perturbação, fazendo com que seja reduzido o número de casos de teste que são gerados, conseqüentemente aumentando a eficiência do mesmo.

Apesar do bom desempenho, o operador *Unauthorized (U)* foi proposto para detectar defeitos relacionados à segurança da informação, como acesso não autorizado ao sistema por um usuário não habilitado. Contudo, tal operador não revelou nenhum defeito desta natureza no sistema.

O operador de *Inversão* (IN) foi capaz de revelar 2 defeitos, contudo, tais defeitos só foram descobertos porque a mensagem SOAP que deu origem aos casos de teste já era, por natureza, um bom caso de teste com alta probabilidade de encontrar defeitos. Este operador foi proposto para revelar defeitos onde a ordem dos elementos em uma mensagem SOAP faz diferença, o que não era o caso dos *Web Services* utilizados, explicando-se a baixa eficiência do mesmo.

Apesar dos resultados obtidos nesta pesquisa, serão necessárias mais evidências de que estes resultados se aplicam a todos os casos. Outras evidências necessárias relacionam-se com os operadores de *Inversão* e *ValorLimite* que se mostraram ineficientes neste estudo de caso.

Além dos 49 defeitos encontrados com a utilização dos operadores, foram registrados mais 18 defeitos a partir da criação de casos de testes escritos pelos testadores utilizando a ferramenta para auxiliá-los na execução dos testes, totalizando 67 defeitos registrados: 49 com o apoio dos operadores de perturbação e 18 criados por testadores. Com isso, pode-se concluir que com a utilização das técnicas mostradas neste trabalho, é possível capturar uma variedade de defeitos em momentos iniciais do desenvolvimento de *Web Services* sem que o esforço com planejamento de casos de teste seja muito intenso. Contudo, em iterações posteriores de testes no mesmo projeto e com a utilização de novas mensagens SOAP que serviram como base para perturbação, verificou-se uma queda acentuada na eficiência do uso de todos os operadores de perturbação aqui relacionados.

Concluindo, este experimento mostrou que a técnica de perturbação de mensagens é eficiente para testes de *Web Services,* uma vez que foram encontrados 73% de todos os defeitos registrados. Quanto aos operadores de perturbação, foi verificado que estes se complementam, não havendo conclusões sobre qual deles é melhor quando aplicados a diversos outros tipos de cenários ou aplicações.

"Se você não consegue pensar
em mais melhorias para o seu *software*,
é possível que tenha chegado ao limite da sua
criatividade e não da qualidade".

7. Conclusões e Trabalhos Futuros

O processo de teste é fundamental para a Garantia de Qualidade do Produto, assim como contribui indiretamente para a melhoria na Garantia de Qualidade do Processo. Contudo, é um procedimento que demanda uma grande quantidade de recursos quando comparado às demais atividades que integram o ciclo de desenvolvimento de *software*. Com isso, é de fundamental importância a utilização de ferramentas que auxiliem na automatização do processo, dando fluidez e agilidade na obtenção dos resultados.

Neste trabalho foi introduzido um grupo de operadores de perturbação de mensagens SOAP utilizados para gerar casos de teste para *Web Services*. Também foi descrita e implementada uma ferramenta, chamada SMAT-WS, que auxilia na utilização destes operadores de perturbação e de outros operadores introduzidos em trabalhos da literatura, assim como em todo o procedimento de testes de *Web Services.*

A ferramenta implementada funciona em ambos os estilos de passagem de mensagens: RPC ou orientado a documentos. Todos os tipos de teste apoiados pela ferramenta baseiam-se na utilização da interação Ponto-a-Ponto. Além dos operadores de perturbação, também fazem parte da ferramenta de teste um módulo consumidor de *Web Services*, um módulo interpretador de WSDL, funcionalidades de teste de regressão e facilidades para o testador, no sentido de diminuir o tempo gasto no procedimento de teste.

Alguns resultados empíricos foram apresentados a partir de um estudo de caso utilizado para validar os operadores propostos. Os resultados foram coletados de um ambiente real de produção de *Web Services*, o que trouxe mais informações empíricas sobre os defeitos mais freqüentes ocorridos no desenvolvimento de *Web Services*.

O grupo de operadores estendidos apresentados nesta pesquisa consiste em um grupo de transformações SOAP que são regras responsáveis por criar as mensagens SOAP modificadas (casos de teste) a partir de uma mensagem SOAP original já existente, havendo independência do processo em relação à utilização de XML *Schemas*. Esta pesquisa mostrou que a técnica de perturbação de mensagens é eficiente quando aplicada a testes de *Web Services,* considerando que grande parte dos defeitos podem ser revelados em momentos iniciais no desenvolvimento de *software*, dispensando grandes demandas por recursos para planejamento e projeto de casos de teste, conseqüentemente causando uma redução nos custos desta atividade. Conclui-se também que os operadores de perturbação são complementares, não havendo evidências, porém, de quais são melhores e mais eficientes para todos os tipos de aplicações.

É conhecido que um suporte automatizado para oráculos é importante para definir a efetividade e custos de um procedimento de testes [MEM03]. Dessa forma, uma estratégia para automatização do oráculo para ferramenta demonstrada é fundamental, uma vez que o número de casos de teste cresce proporcionalmente ao número de operadores de perturbação utilizados no procedimento, o que se caracteriza como uma desvantagem desta técnica.

7.1. Trabalhos Futuros

Algumas oportunidades de trabalhos futuros foram vislumbradas após a conclusão deste trabalho. Algumas delas envolvem a teoria de perturbação de mensagens, a ferramenta para testes de *Web Services* e também algumas conclusões tiradas com os resultados do experimento.

Foi pesquisada apenas a perturbação de mensagens SOAP dentro da tecnologia de *Web Services*, apesar de que, a especificação WSDL contém um grupo de informações que podem ser interessantes na derivação de outro grupo de operadores.

Alguns outros estudos e experiências sobre os defeitos mais freqüentes ocorridos na produção de *Web Services* podem ser utilizados como base para

propostas de novos operadores de perturbação, assim como avaliação dos operadores propostos neste trabalho.

Algumas novas extensões para a ferramenta podem ser implementadas e experimentadas para atender novas necessidades de teste para *Web Services*, como:

- Suporte para interação Multilateral entre *Web Services* e suas novas necessidades de cenários de teste;
- Implementação de um provedor de serviços que emula as funcionalidades de um *Web Service* real;
- Implementação de facilidades para permitir que interações múltiplas entre *Web Services* sejam contempladas na ferramenta com o auxílio de novas especificações como *Business Process Execution Language* (BPEL) [BPL03], por exemplo;
- Implementação de outros operadores de perturbação de mensagens;
- Implementação de um oráculo inteligente capaz de decidir sobre a corretude das respostas obtidas para os casos de teste executados, uma vez que os operadores de perturbação tendem a gerar um número elevado de casos de teste.

Outro ponto a ser examinado com mais detalhes é a possibilidade de construir aplicações baseadas na tecnologia de *Web Services* em diversas plataformas, uma vez que foram verificados neste trabalho alguns problemas de integração entre *frameworks* de desenvolvimento. Como mais uma proposição de estudos futuros, estes problemas de integração percebidos podem ser utilizados para gerarem casos de teste para verificar conformidade com alguns padrões de *Web Service*.

Referências

[AX00] **Apache Axis.** The Apache Software Foundation – 2000. http://ws.apache.org/axis/ (acessado em Janeiro 2006).

[BEI90] Beizer, B. **Software Testing Techniques.** Van Nostrand Reinhold, Inc, New York NY, 2nd edition, 1990. ISBN 0-442-20672-0.

[BLO02a] Bloomberg, J. **Web Services Testing: Beyond SOAP.** ZapThink LLC, special to SearchWebServices, Setembro 2002.

[BLO02b] Bloomberg, J. **Testing Web Services Today and Tomorrow.** The Rational Edge E-zine for the Rational Community – 2002 http://www.therationaledge.com, capturado em Fevereiro 2005.

[BPL03] **BPEL Specification**, version 1.1 – Maio 2003. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/ (acessado em Janeiro 2006).

[BUD81] Budd, T. A. **Mutation Analysis: Ideas, Examples, Problems and Prospects, Computer Program Testing.** North-Holand Publishing Company - 1981, pp. 129-148.

[BUL03] Bultan, T.; Fu, X.; Hull, R. and Su, J. **Conversation specification: A new approach to design and analysis of e-service composition.** In The Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary, Maio 20-24 2003.

[CER02] Cerami, E. **Web Services Essentials. Distributed Applications with XML-RPC, SOAP, UDDI and WSDL.** Editora O'Really – 2002.

[DAV02] Davidson, N. **Web Services Testing.** The Red-gate software technical papers, 2002. http://www.redgate.com.testing.

[DEM78] De Millo, R. A. **Hints on Test Data Selection: Help for the Practicing Programmer.** IEEE Computer, 11(4):34-41, Abril 1978.

[DEU79] Deutsch, M. **Verification and Validation.** In Software Engineering (R. Jensen and C. Tonies, eds.), Prentice-Hall, 1979, pp. 329-408.

[DIL05] Di Lucca, G. **Testing Web-Based Applications: the State of the Art and Future Trends.** In QATWBA'05 - Workshop of the International Computer Software and Applications Conference, COMPSAC 2005 page 65. IEEE Press, Edinburgh- Scotland, June 2005.

[DOM04] **W3C. Document Object Model.** 2004, capturado em Fevereiro 2005 http://www.w3c.org/DOM**.**

[DUNS05] **Data Universal Numbering System.** http://www.dnb.com/english/duns, capturado em Abril 2005.

[FRA88] Frankl, P.; Weyuker, E. J. **An Applicable Family of Data Flow Testing Criteria.** IEEE Transactions on Software Engineering, vol. 14, no. 10, October 1988, pp. 1483-1497.

[GAB02] Gabrick, K.; Weiss, D. **J2EE and XML Development.** Manning Publications – 2002.

[HEC04] Heckel, R.; Lohmann, M. **Towards Contract-Based Testing of Web Services.** Electronic Notes in Theoretical Computer Science 82 No. 6 (2004) URL: http://www.elsevier.nl/locate/entcs/volume82.html**.** 12 páginas.

[HOR06] Horstmann, M.; Kirtland, M. **DCOM Architecture.** Microsoft Developer Network http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomarch.asp (acessado em Janeiro 2006).

[IEEE90] Institute of Electrical and Electronics Engineers. **IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.** New York, NY: 1990.

[KRE01] Kreger, H. **Web Service Conceptual Architecture (WSCA 1.0).** IBM Software Group – 2001.

[LEE01] Lee, S. C.; Offutt, J. **Generating Test Cases for XML-based Web Component Interactions Using Mutation Analysis.** Software Reliability Engineering Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE'01), p.200, November 27-30, 2001.

[MAL91] Maldonado, J. C. **Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software.** Tese de Doutorado, DCA/FEE/UNICAMP. Campinas, SP – 1991.

[MCC76] McCabe, T. **A Software Complexity Measure.** IEEE Transactions on Software Engineering, Vol. 2, pp. 308-320, 1976.

[MEM03] Memon, A.; Banerjee, I.; Nagarajan, A. **What Test Oracle Should I Use for Effective GUI Testing?**, ase, p. 164, 18th IEEE International Conference on Automated Software Engineering (ASE'03), 2003.

[MUR02] Murray, PJ.; Golluscion, E. **CORBA and Web Services.** Cape Clear Software – 2002 http://www.omg.org/news/whitepapers/CORBA-Web%20Services.pdf (acessado em Janeiro 2006).

[MYE79] Myers, G. **The Art of Software Testing.** Wiley – 1979.

[NAM99] W3C. **Namespaces in XML**. Janeiro de 1999, capturado em Janeiro de 2006 - http://www.w3.org/TR/REC-xml-names/.

[NEW02] Newcomer, E. **Understanding Web Services: XML, WSDL, SOAP and UDDI.** Addison-Wesley – 2002. ISBN 0-201-75081-3.

[OFF02] Offutt, J. **Quality Attributes of Web Software Applications.** IEEE Software: Special Issue on Software Engineering of Internet Software, Vol. 19, No. 2, pp. 25–32, 2002.

[OFF04] Offutt, J.; Xu, W. **Generating Test Cases for Web Services Using Data Perturbation.** ACM SIGSOFT Software Engineering Notes, v.29 n.5, Setembro 2004.

[ORG06] **CORBA Specification.** Object Management Group. http://www.omg.org/ (acessado em Janeiro 2006).

[PRES02] Pressman, R. **Engenharia de Software.** McGraw Hill, 5ª edição – 2002.

[RAP85] Rapps, S.; Weyuker, E. J. **Selecting Software Test Data Using Data Flow Information.** IEEE Transactions on Software Engineering, Vol. 11, No 4, pp. 367-375 – 1985.

[REY06] Reynolds, M. **Web Services 101. What are Web Services?** http://www.webservicesarchitect.com/content/articles/reynolds01.asp**,** capturado em Janeiro 2006.

[ROC01] Rocha, A. R. C.; Maldonado, J. C.; Weber, K. C. **Qualidade de Software. Teoria e Prática.** Prentice Hall – 2001.

[SPE02] Spett, K. **SQL injection: Are your Web applications vulnerable?** White paper in SPI Dynamics, Inc., 2002. http://www.spidynamics.com/whitepapers/ WhitepaperSQLInjection.pdf (acessado em Janeiro 2006).

[SUN06] **Java Remote Method Invocation (JRMI).** Sun Developer Network. http://java.sun.com/products/jdk/rmi/ (acessado em Janeiro 2006).

[TSA02] Tsai, W. T.; Paul, R., Song, W. and Cao, Z. **Coyote: An XML-based framework for Web services testing.** In 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02), Tokyo, Japan, Outubro 2002.

[TSA04] Tsai, W.; Chen, Y.; Cao, Z.; Bai, X.; Huang, H.; Paul, R. **Testing Web Services Using Progressive Group Testing.** AWCC 2004, LNCS 3309, pp. 314–322, 2004.

[W3C] **World Wide Web Consortium.** http://www.w3.org, capturado em Janeiro 2006.

[WSA04] W3C Working Group Note. **Web Service Architecture.** Fevereiro - 2004, capturado em Fevereiro 2005 - http://www.w3.org/TR/ws-arch.

[WSDL01] W3C Note. **Web Services Description Language (WSDL).** Março 2001, capturado em Fevereiro 2005 - http://www.w3.org/TR/wsdl**.**

[WUO02] Wu, Y.; Offutt, J. **Modeling and Testing Web-Based Applications.** GMU ISE Technical ISE-TR-02-08, Novembro 2002.

[WXML] W3C. **XML Recommendation.** Maio 2001, capturado em Março 2005 - http://www.w3.org/xml.

[XS01] W3C Recommendation. **XML Schema.** Outubro 2004, capturado em Fevereiro 2005 - http://www.w3.org/TR/xmlschema-0/.

[XUO05] Xu, W.; Offutt, J.; Luo, J. **Testing Web Services by XML Perturbation.** Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), 2005.

Apêndice A – Telas da ferramenta de testes SMAT-WS.

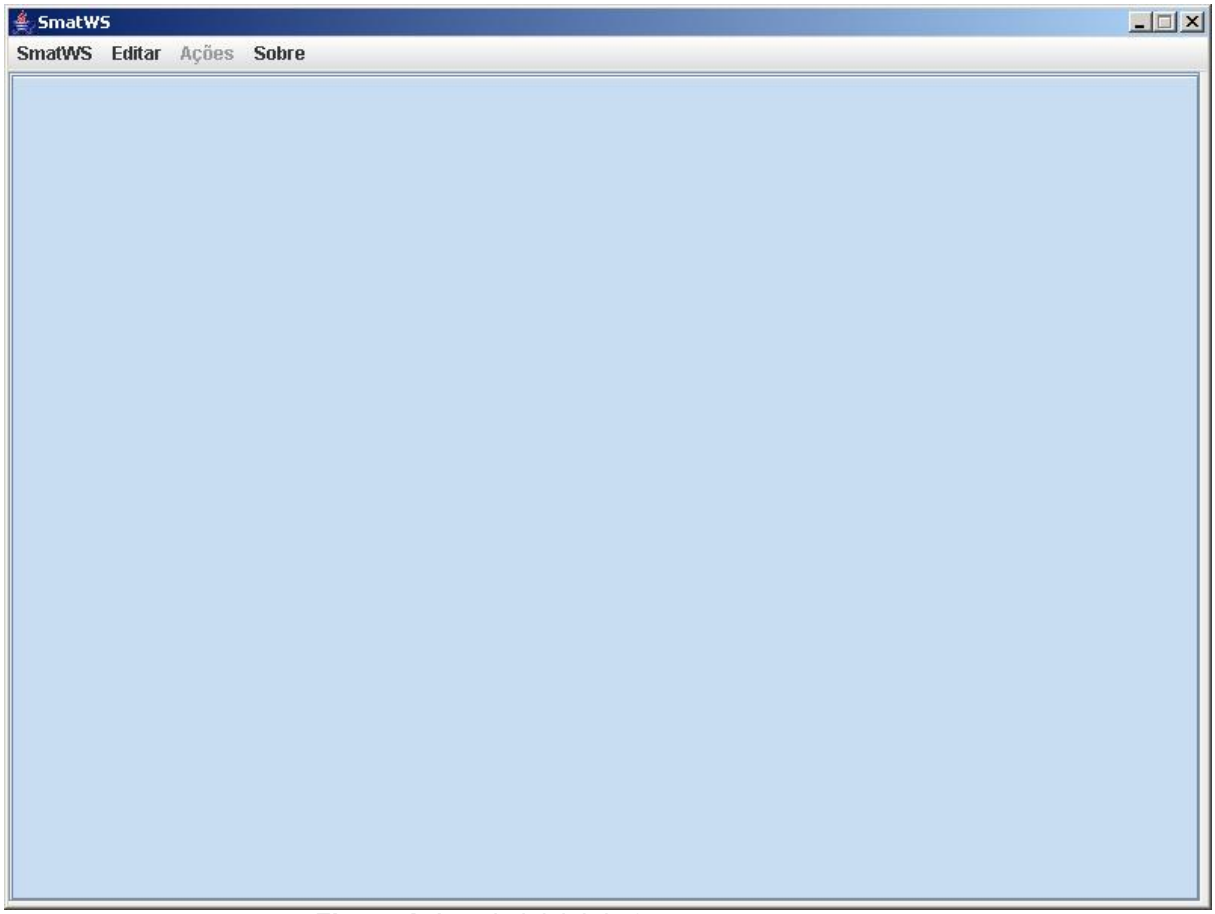

Figura A.1: Tela inicial da ferramenta SMAT-WS.

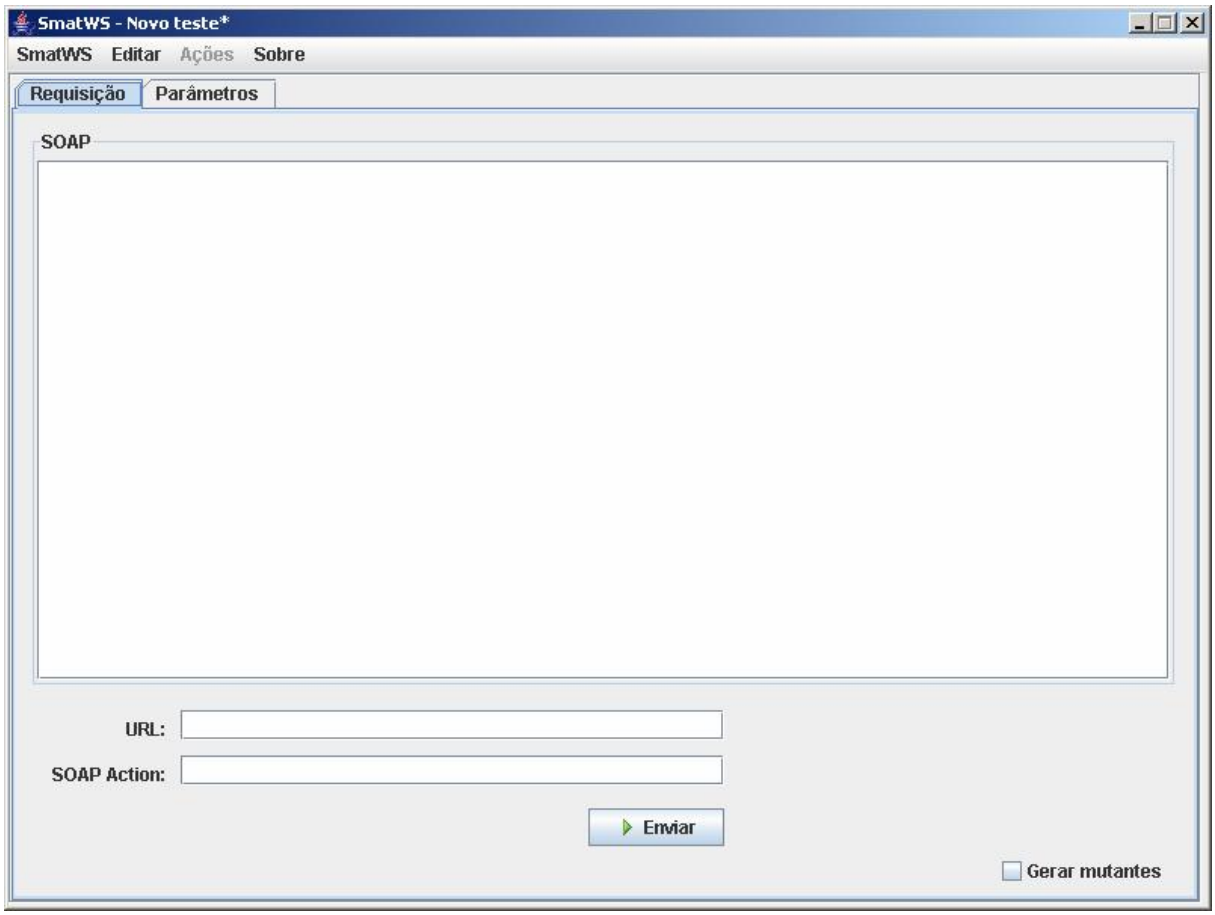

Figura A.2: Inserindo uma mensagem SOAP a ser enviada.

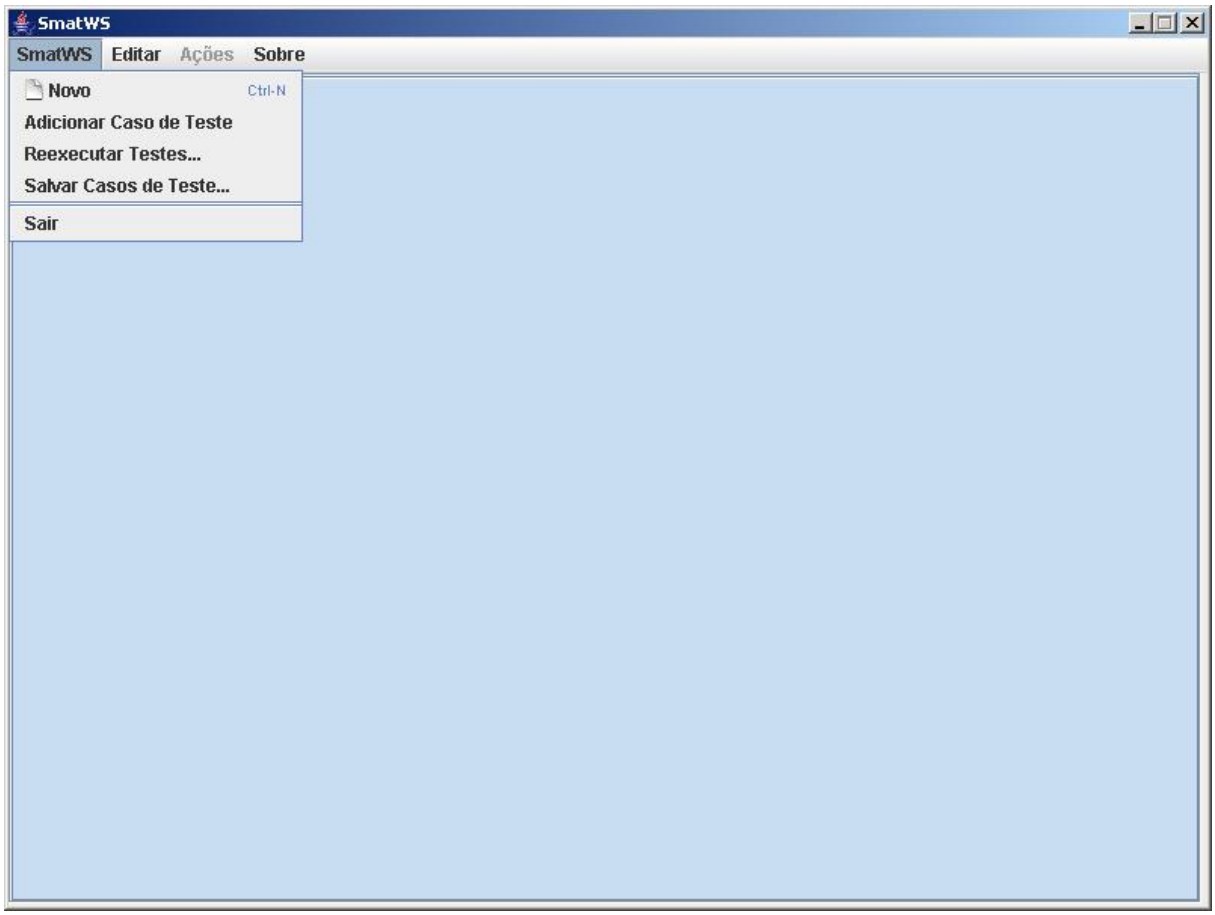

Figura A.3.: Menu principal da ferramenta.

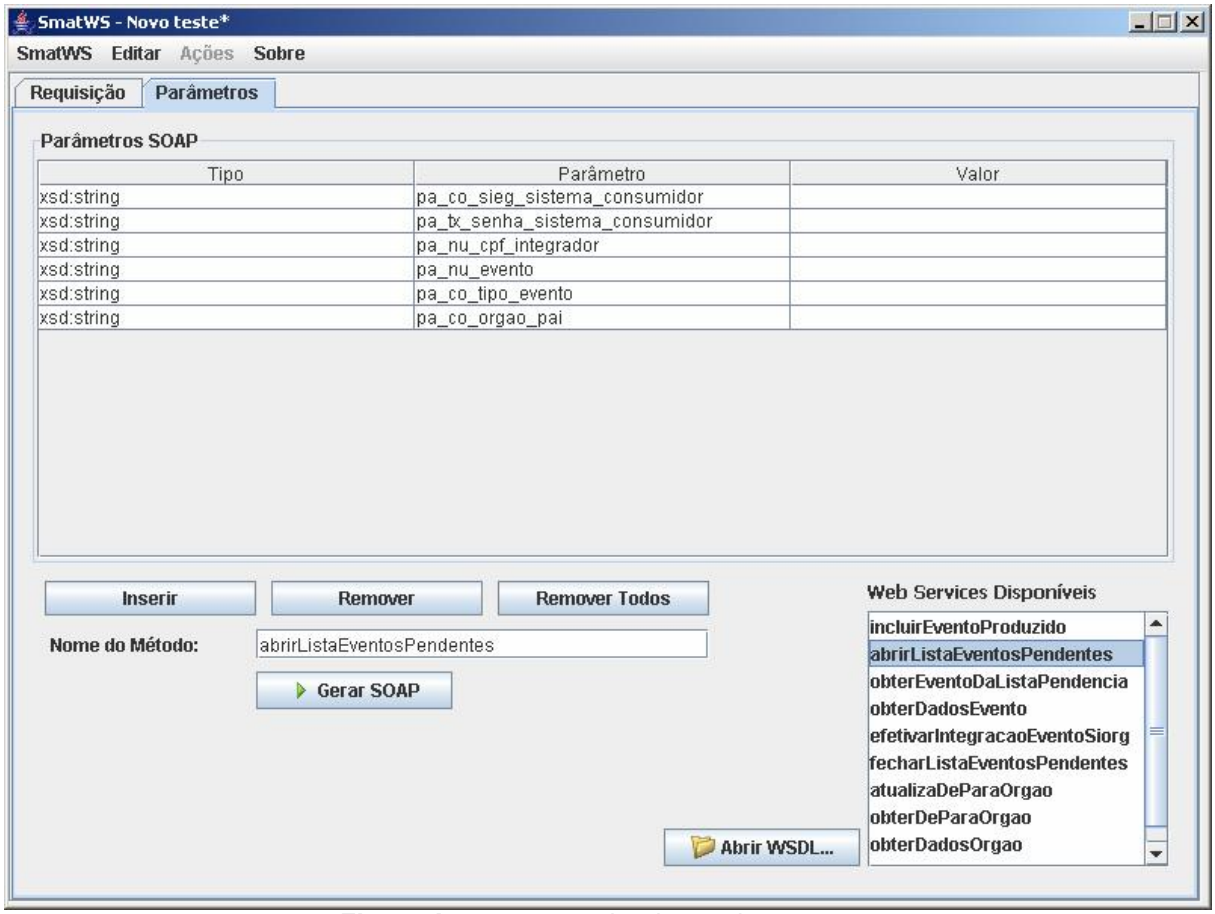

Figura A.4: Interpretador de arquivos WSDL.

Anexo I – Exemplo de WSDL

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol" type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
              <complexType>
                  <all>
                      <element name="price" type="float"/>
                  </all>
              </complexType>
           </element>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
        </operation>
    </binding>
```

```
    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```